МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ПУЛЮЯ

Кафедра комп'ютерних наук

КОНСПЕКТ ЛЕКЦІЙ
з дисципліни

«РОЗПОДІЛЕНІ СИСТЕМИ МОНІТОРИНГУ ТА КЕРУВАННЯ»

для студентів освітнього рівня «бакалавр»
спеціальності 125 «Кібербезпека»

Тернопіль
2016

УДК 681.3
ББК 32.97
К65

Укладачі:
Шимчук Г.В., асистент,
Маєвський О.В., ст. викладач,
Назаревич О.Б., канд. техн. наук, асистент.

Рецензент:
М.М. Касянчук, канд. фіз.-мат. наук, доцент.

Методичні вказівки розглянуто й затверджено на засіданні
кафедри компютерних наук
Тернопільського національного технічного університету імені Івана Пулюя
протокол № 2 від 09 вересня 2015 р.

Схвалено та рекомендовано до друку на засіданні методичної комісії
факультету комп'ютерно-інформаційних систем та програмної інженерії
Тернопільського національного технічного університету імені Івана Пулюя
протокол № 2 від 25 вересня 2015 р.

к65 Конспект лекцій з дисципліни «Розподілені системи моніторингу та керування» для студентів освітнього рівня «бакалавр» спеціальності 125 – «Кібербезпека» / Укладачі : Шимчук Г.В., Маєвський О.В., Назаревич О.Б. – Тернопіль : Вид-во ТНТУ імені Івана Пулюя, 2016 – 316 с.

УДК 681.3
ББК 32.97

Конспект лекцій призначений для полегшення засвоєння дисципліни "Розподілені системи моніторингу та керування". Складається з урахуванням модульної системи навчання.

Вказівки складені з урахуванням матеріалів літературних джерел, названих у списку.

Відповідальний за випуск: *М.В. Приймак*, докт. техн. наук, професор.

ЗМІСТ

ЛЕКЦІЯ 1. ОСОБЛИВОСТІ РОЗПОДІЛУ ЗАДАЧ І ПЕРЕДАЧІ ДАНИХ. ІСТОРІЯ ВИНИКНЕННЯ ТА ЕВОЛЮЦІЯ GRID-СИСТЕМ. КЛАСИФІКАЦІЯ GRID-СИСТЕМ.

1.1. Особливості розподілу задач і передачі даних

Останні десять років є роками зародження та розвитку нового напрямку в інформаційних технологіях, назву якому (як традиційно вважається) дали у 1998 році Я. Фостер та К. Кессельман – «Грід» (англ. «Grid») [1]. Grid як засіб сумісного використання обчислювальних потужностей та сховищ даних дозволяє вийти за межі простого обміну даними між комп'ютерами і, зрештою, перетворити їх глобальну мережу на свого роду гігантський віртуальний комп'ютер, доступний у режимі віддаленого доступу з будь-якої точки, незалежно від місця розташування користувача.

Ідея використовувати мережу суперкомп'ютерів для вирішення задач, мабуть, зародилася набагато раніше (спроби робилися з 60-х років XX століття), однак зараз вона набула завершеної форми «концепції Grid». Традиційно, причетними до розвитку Grid-обчислень вважають фізиків-ядерщиків – і дотепер їх потреба в обробці колосальних об'ємів дослідних даних є рушійною силою для реалізації програм по впровадженню Grid, згадати хоча б діяльність Європейського центру ядерних досліджень (CERN). Однак Grid має потенційно велику кількість й інших областей застосування, оскільки пропонує універсальний підхід до розв'язку проблеми нестачі обчислювальних ресурсів – адже очевидно, що в загальному випадку мережа суперкомп'ютерів є спроможною вирішити складніші задачі, ніж кожен з її складових вузлів окремо.

Якщо перекладати дослівно, Grid означає «ґрати». Погодьтеся, асоціації, пов'язані в нашій мові з цим словом, зовсім не відповідають змісту вільної кооперації комп'ютерів для високопродуктивних обчислень, закладеному в технологіях Grid. Найближче за смислом, мабуть, поняття power grid – мережа електроживлення, розподілений ресурс загального користування, коли кожен може легко під'єднатися через розетку і використовувати стільки електроенергії, скільки йому потрібно. Аналогічно користувачі з допомогою Grid отримують можливість прямого підключення до віддаленої обчислювальної мережі, не цікавлячись, звідки беруться потрібні для роботи обчислювальні ресурси й дані, які для цього використовуються лінії передачі, паролі чи протоколи тощо. При цьому аналогом інфраструктури електричних мереж (ліній електропередачі, підстанцій, трансформаторів, диспетчерських пунктів та ін.) виступає Grid –інфраструктура з програмним Grid забезпеченням (ПГЗ), з допомогою якого виконується „віртуалізація" ресурсів. Розвиток ПГЗ починався від базових засобів, що підтримують дистанційний доступ до ресурсів, пройшов стадію окремих систем, їх пакетів і привів до створення платформ – взаємноузгоджених наборів засобів, здатних дати комплексне рішення завдання обслуговування Grid-інфраструктури виробничого призначення.

Проте слід зазначити, що хоча динаміка розвитку в цій галузі дозволяє сподіватися на успішне якнайширше впровадження Grid – технологій у найближчому майбутньому, багато питань все ще є предметом дискусій і досліджень. Нижче коротко перераховуються та пояснюються основні властивості Grid, до яких відносять:

– Спільне використання розподілених ресурсів. Воно відкриває можливості з співробітництва, яких були б важко досягти іншими засобами. Водночас виникають питання «справедливості» розподілу ресурсів, створення та управління віртуальними організаціями (ВО).

– Об‘єднання потужностей. Таким чином будується система з сумарною потенційною обчислювальною потужністю, що перевершує потужності її складових, і при цьому досягається більш ефективне використання апаратних засобів (зменшується простоювання). Постають жорсткі вимоги до каналів зв‘язку.

– Віртуалізація. Включає в себе такі поняття, як: приховування (маскування) від користувача складності апаратної та програмної реалізації системи та її складових, географічних відстаней між вузлами, приналежності вузлів різним організаціям, створення ілюзії роботи з «віртуальним суперкомп‘ютером».

– Неоднорідність (гетерогенність). Типовий Grid складається з множини різнорідних апаратних засобів та різноманітного програмного забезпечення (і має успішно функціонувати в таких умовах).

– Децентралізоване управління. Немає одного єдиного «власника» всієї системи, що вимагає використання механізмів розподіленого управління.

– Інтероперабельність. Функціональна сумісність роботи різних компонентів Grid та навіть різних Grid -інфраструктур базується на стандартизації інтерфейсів. Підходи, що не враховують стандартів, є мало перспективними.

– Прозорість доступу. Grid має надавати доступ до ресурсів системи користувачам, не зважаючи на конкретну топологію мережі чи локальну реалізацію механізмів доступу до тих чи інших вузлів та їх ресурсів.

– Масштабованість. Grid має забезпечувати механізми включення нових джерел ресурсів, користувачів, сховищ даних тощо без впливу на існуючих учасників: Grid повинен мати здатність динамічно реконфігуруватись.

– Безпека. Одна з головних вимог до Grid – системи – безпека доступу до ресурсів, що зумовлює обмежений набір дозволених операцій у авторизованих користувачів та програм.

Розподілені обчислювальні технології, що існували до появи Grid - технологій, не забезпечують виконання представлених вище вимог. Наприклад, Інтернет-технології спрямовані на комунікаційний і інформаційний обмін між комп'ютерами, але не надають інтегрованого підходу для координованого використання ресурсів на декількох обчислювальних системах. Бізнес-технології фокусуються лише на проблемах спільного використання інформації, найчастіше через централізовані сервери. Розподілені

обчислювальні технології масштабу підприємства, такі як CORBA або Enterprise Java, дозволяють розділяти ресурси тільки в рамках однієї організації. Засоби Open Group's Distributed Computing Environment (DCE) підтримують захищене спільне використання ресурсів декількох сайтів, однак більшість віртуальних організацій знаходить ці інструменти обтяжними й негнучкими. Провайдери сервісів зберігання даних (storage service provider – SSP) і сервісів додатків (application service provider – ASP) допускають зберігання даних і виконання обчислень на третій стороні, але з деякими обмеженнями (наприклад, SSP ресурси доступні покупцям лише через приватні віртуальні мережі (virtual private network – VPN). Таким чином, нинішні технології або не дозволяють об'єднати різноманітні типи ресурсів, або не надають гнучкості в керуванні поділюваними зв'язками.

Важливо підкреслити, говорячи про використання Grid-технологій, що вони дозволяють не тільки вирішувати наукові й практичні завдання, раніше недоступні через занадто великий час, потрібний для одержання відповіді. Grid створює основу для нової організації науки, високотехнологічного виробництва, соціального життя, дозволяє більш ефективно та надійно управляти ресурсами суспільства.

Концепція Grid базується на наступних основоположних моментах:

– швидке та постійне збільшення продуктивності мікропроцесорів масового виробництва (сучасний персональний комп'ютер на базі процесора може зрівнятися за швидкістю обчислень із суперкомп'ютерами 10-літньої давнини);

– поява швидкісних оптоволоконних ліній зв'язку – сьогодні в розвинених країнах базові лінії зв'язку в мережі Інтернет мають пропускну здатність 10 Гб/с і вище, а підключення до мережі багатьох наукових організацій відбувається на швидкості в 1-2 Гб/с;

– феномен Інтернету, глобалізація процесу обміну інформацією й інтеграції світової економіки;

– безперервне вдосконалювання технологій і засобів інформаційної безпеки.

По суті, Grid є «надбудовою» над Інтернетом. Її основне призначення – організація розподілених обчислень для рішення серйозних задач науки й технології, які вимагають більших обчислювальних ресурсів, а саме, потужності комп'ютерів, ресурсів зберігання даних, часу обчислень. На відміну від безструктурної мережі Інтернет, Grid – чітко впорядкована система. Певною мірою Grid можна умовно назвати обчислювальним Інтернетом. Користувач, підключаючись до Grid, одержує доступ до потужності тисяч машин, на яких він може здійснювати обчислення й зберігати як величезні масиви даних, так і інформацію, отриману в результаті їхньої обробки. У цій мережі приділяється першорядна увага проблемам безпеки – адже анонімність, зручна при спілкуванні в Інтернеті, може стати надзвичайно небезпечної при роботі з науковими або практичними даними. Запитуючи яку-небудь інформацію в глобальній базі даних Grid, користувач одержить вичерпну відповідь на питання про її достовірність, повноту й доступність.

Відомо, що зараз у світі близько 500 млн. персональних комп'ютерів, однак, у середньому використовується всього лише 15% обчислювальних ресурсів! Система Grid-обчислень дозволить більш ефективно використати наявну базу обчислювальних ресурсів.

У комп'ютерних Grid-системах різні організації, що мають загальні наукові або практичні інтереси, на добровільній основі створюють об'єднання, що у Grid-технологіях називається віртуальною організацією(ВО). Учасники віртуальної організації зв'язані між собою за допомогою Інтернету таким чином, що їхні обчислювальні потужності поєднуються. Система містить у собі обчислювальні ресурси й ресурси зберігання даних, але при цьому кожна організація контролює використання своїх ресурсів. Користувачі можуть одержувати практично необмежені ресурси для обчислень і зберігання даних, не замислюючись про їхнє походження. Кожний з учасників надає свої обчислювальні ресурси (або їхню частину) для використання іншими учасниками, і в той же час отримує доступ до ресурсів інших учасників.

Розвиток Grid-технологій стає стратегічним національним завданням. Дуже незабаром країни, що не мають розвинутої Grid-інфраструктури, навіть не зможуть претендувати на статус розвинутої країни.

1.2. Історія виникнення та еволюція GRID-систем

Напевно, краще почати з тих десятиліть інтенсивних досліджень, розробок і впровадження апаратного і програмного забезпечення та додатків для паралельного обчислення на багатьох комп'ютерах, що відбувалися в 1980-х роках Паралельне обчислення в 1980-х роках зосередило зусилля дослідників на розвитку алгоритмів, програм і архітектур, які підтримували одночасну роботу кількох процесорів. Оскільки розробники додатків почали створювати широкомасштабні програми, що вимагали ресурсів набагато більше, ніж можуть забезпечити навіть найшвидші паралельні комп'ютери, деякі групи дослідників почали розглядати розподілення ресурсів за межами основного комп’ютера як спосіб рішення задач все більшого і більшого масштабу.

Впродовж 1980-х і 1990-х, програмне забезпечення для паралельних комп'ютерів зосередилося на тому, щоб забезпечити могутні механізми для управління зв'язками між процесорами, і розробці і створенні середовищ для паралельних машин. Паралельна віртуальна машина (Parallel Virtual Machine, PVM), інтерфейс передачі повідомлень (Message Passing Interface, MPI), високопродуктивний ФОРТРАН (High Performance Fortran, HPF) і OpenMP були розроблені з ціллю підтримки зв'язку для додатків, що масштабуються (scalable applications) [2]. Були розроблені успішні прикладні парадигми, щоб привести в дію величезний потенціал розділеної і розподіленої архітектури пам'яті. Спочатку вважали, що Grid буде найбільш корисною в розширенні парадигм паралельного обчислення від тісно зв'язаних груп до географічно розподілених систем. Проте на практиці Grid використовувалася більше як платформа для інтеграції вільно зв'язаних додатків (деякі компоненти яких могли б працювати на паралельній машині), і для зв'язку розподілених ресурсів

(зберігання, обчислення, візуалізація, інструменти). Фундаментальна задача Grid управління цими різнорідними компонентами замінює собою задачу твердої синхронізації типово ідентичних компонентів (як в моделі SPMD, одна програма, численні дані – single program multiple data).

Впродовж 1980-х дослідники з різних областей також почали об'єднуватися, щоб вирішити проблеми «Великого Виклику» («Grand Challenge») [66], тобто ключові проблеми в науці і інженерії, для яких великомасштабна обчислювальна інфраструктура забезпечила би фундаментальний інструмент, щоб досягти нових наукових відкриттів. Команди «Великого Виклику» та міждисциплінарні команди розробили модель для співпраці, і ця модель дуже сильно вплинула на те, яким чином великомасштабна наука ведеться сьогодні. Нині міждисциплінарні дослідження не тільки забезпечили модель для співпраці, але також надихнули цілі дисципліни (наприклад, біоінформатику), на інтеграцію раніше розосереджених областей науки.

Проблеми, зв'язані з проведенням міждисциплінарних і, часто, географічно розрізнених співробітництв, дослідники вирішують як з координацією, так і з розподілом робіт – двома фундаментальними поняттями в Grid-обчисленні. У 1990-х роках американська випробувальна програма Gigabit [5] звернула головну увагу на розподілені регіональні та глобальні додатки. Кожна з випробувальних моделей – Aurora, Blanca, Casa, Nectar і Vistanet – була розроблена з подвійними цілями: по-перше, досліджувати потенційну архітектуру мережі випробувальної моделі, а по-друге – вивчити їх корисність для кінцевих користувачів. У цій другій меті кожна випробувальна модель забезпечила площу для експериментування з розподіленими додатками.

Першим сучасним Grid зазвичай вважають проект I-WAY (Information Wide-Area Year), розроблений як експериментальний демонстраційний проект у 1995 році, який об’єднав в національну розподілену експериментальну мережу 17 обчислювальних вузлів, зв'язаних за допомогою високошвидкісної магістралі на основі технології ATM (1.5 Мб/с – 9.6 Мб/с) [6]. Було впроваджено розподілену файлову систему і брокер ресурсів, який містив центральний вузол, що керував мережею, та агентів на решті вузлів. Тоді було розроблено також 60 додатків і упроваджено в I-WAY, Розробка інфраструктури і додатків для I-WAY забезпечила конструктивний і могутній досвід для першого покоління сучасних дослідників і проектантів Grid.

У пізніх 1990-х розробники Grid зібралися на форумі Grid, який згодом розвинувся в Глобальний форум Grid (Global Grid Forum, GGF) [12] і який став результати ранніх досліджень використовувати для розроблення стандартів для майбутніх Grid. З тої пори дослідження в галузі Grid технологій стали глобальними: в них співпрацюють багато дослідників із Сполучених Штатів, Європи і Азіатсько-тихоокеанського регіону. Фондові агентства, комерційні кредитори, академічні установи, національні центри і лабораторії об'єднали свої зусилля, щоб сформувати ерудоване співтовариство з високою зацікавленістю в побудові Grid. Більш того, дослідження в суміжних галузях, таких як

обчислювальні мережі, однорангові обчислення тощо, генерують додаткові ідеї, що втілюються в Grid.

Сьогодні виділяють три стадії еволюції Grid: системи першого покоління, які були попередниками того Grid, що ми маємо сьогодні; системи другого покоління, які розвивали програмне Grid забезпечення, щоб оперувати великомасштабними даними та обчисленнями; і системи третього покоління, де акцент переходить до глобального розподіленого співробітництва, до сервісно-орієнтованого підходу і обробки величезних об'ємів даних (рис.1.1).

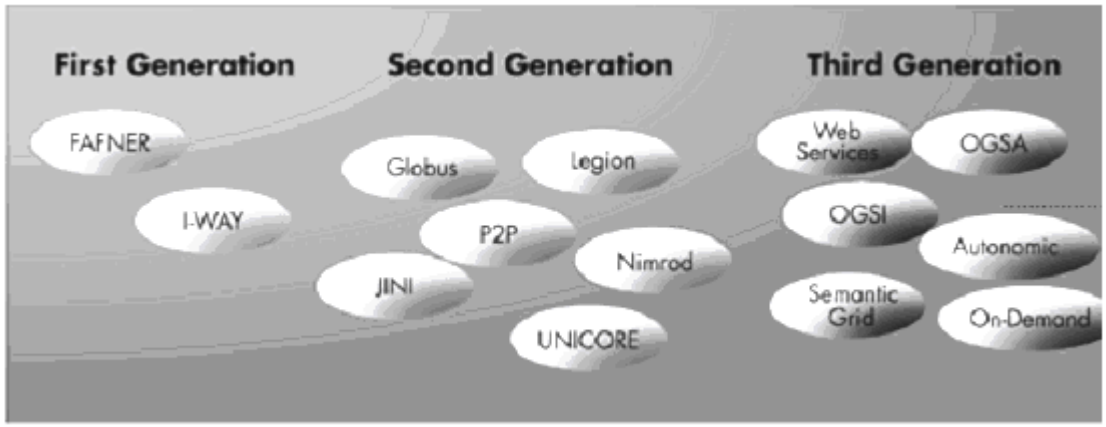


Рисунок 1.1. – Покоління Grid: системи

Еволюція Grid відображає стрімку еволюцію інформаційних технологій, коли потужність процесорів, що визначається кількістю транзисторів на чипі, згідно закону Мура подвоюється кожні 18 місяців; щільність зберігання даних подвоюється кожні 12 місяців, а пропускна спроможність мереж збільшується вдвічі всього за 9 місяців. Тому сьогодні в розпорядженні вчених і інженерів є більш ніж 100 млн. хостів Інтернету, високошвидкісні мережі з пропускною спроможністю більш ніж 10 Гб/с; засоби зберігання даних з ємністю петабайти і обчислювальні потужності з продуктивністю більш ніж 100 Тф/с. Звичайно, еволюція – це безперервний процес, і не завжди чітко можна виділити точні межі, але такий еволюційний поділ достатньо описує проблеми, що виникали перед розвитком Grid на кожному етапі.

1.3. Еволюція GRID: перше покоління (1990-1996 роки)

Перші проекти Grid були спробами зв'язати суперобчислювальні вузли (сайти); у той час цей підхід був відомий як метаобчислення. Метакомп'ютер – це багатопроцесорна система, роль системної шини в якій виконує Інтернет, що зв'язує безліч вузлів, що мають власні процесори, оперативну і зовнішню пам'ять, пристрої введення/виведення (рис.1.2).

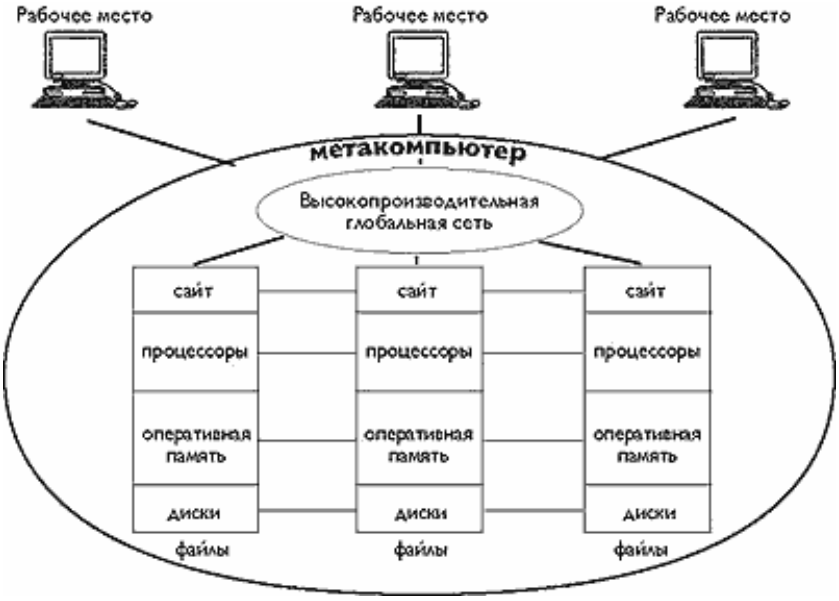


Рисунок 1.2. – Структура метакомп’ютера

Початком цього періоду вважається старт проекту CASA в 1989 році одним з декількох американських іспитових стендів Gigabit [74]. На початку та середині дев'яностих років відзначають появу перших метаобчислень, або середовищ Grid. Як правило, мета цих ранніх метаобчислювальних проектів полягала в тому, щоб забезпечити ряд високопродуктивних додатків обчислювальними ресурсами. Поперед цієї технології йшли два представницьких проекти – FAFNER [68] та I-WAY [6]. Ці проекти відрізняються різними шляхами реалізації, але для ефективної роботи вони повинні були подолати багато подібних перешкод, такі як зв'язок, керування ресурсами та віддаленими даними. Ці два проекти також спробували вирішити проблеми метаобчислення, при цьому FAFNER був здатний до виконання на будь-якій робочій станції, що мала більше ніж 4 мегабайти пам'яті, I-WAY був засобом об'єднання ресурсів величезних американських суперобчислювальних центрів.

FAFNER мав недоліки, які зараз у Grid системах відсутні. Наприклад, кожний клієнт повинен був компілювати, встановлювати і виконати демон FAFNER, щоб здійснювати факторизації. Індивідуальні обчислювальні завдання були нездатні об’єднуватися один з одним, або з батьківським Web-сервером. Аналогічно, I-WAY мав багато таких властивостей, які сьогодні здаються недоречними, наприклад, сервіси I-WAY повинні були бути встановлені окремо на кожному вузлі. Незважаючи на вищезгадані особливості, FAFNER і I-WAY були досить інноваційними та успішними проектами. Кожний проект був в авангарді метаобчислення та допоміг прокладати шлях багатьом проектам Grid з наступного другого покоління.

До здобутків першого покоління Grid систем слід віднести таке:

– перетворення комп'ютера на пристрій, розрахований на багато користувачів, і створення систем розділення часу;

– організацію взаємодії на рівні процесів шляхом:

– організації загального адресного простору;

– розподіленої загальної пам'яті (DSM);

– передачі повідомлень (MPI, PVM) для підтримки гетерогенних систем і роботи в локальній/глобальній мережі;

– перенесення моделі доступу до програмних і апаратних ресурсів комп'ютера на розподілене обчислювальне середовище – користувач метакомп‘ютера може працювати з його ресурсами так само, як з ресурсами власного комп'ютера;

– відпрацювання наступної схеми роботи:

– агент на машині визначає факт простою і зв'язується з сервером, що управляє;

– сервер, що управляє, відправляє агентові нове завдання (дані для обробки, область в просторі пошуку);

– після закінчення обчислень результати відправляються назад на сервер;

– підтримку динамічної конфігурації середовища, коли метакомп'ютер не має постійного складу і динамічно організовується з географічно розподілених ресурсів, тимчасово делегованих їх фактичними власниками (віртуалізація метакомп'ютера);

– організацію однорідного доступу до обчислювальних ресурсів великої кількості комп'ютерів в локальній або глобальній мережі;

– рішення задач, що допускають декомпозицію на велике число невеликих, незалежних під задач.

Піонерські проекти Grid заслуговують на те, щоб бути розглянутими більш детально.

1.3.1. FAFNER

Алгоритм шифрування відкритого ключа RSA був винайдений у лабораторії Масачусетського технологічного інституту інформатики в 1976-77 роках. Цей алгоритм широко використовується, наприклад, у протоколі захищених сокетів (Secure Sockets Layer, SSL). Безпека RSA заснована на твердженні, що дуже важко розкласти на множники надзвичайно великі числа, які складаються із сотень цифр. У березні 1991 року RSA Data Security Inc кинула виклик проблемі факторизації RSA (RSA Factoring Challenge). Факторизація в обчислювальному відношенні є дуже витратною. Через цю причину були розроблені паралельні алгоритми факторизації з метою широкого її використання. Використовувані алгоритми тривіально паралельні та не вимагають ніякого зв'язку після початкової установки, і багато машин можуть забезпечити невеликий внесок кожна у загальне рішення завдання по факторизації. Ранні зусилля ґрунтувалися на обміні інформацією та одержання коду факторизації по електронній пошті. В 1995 році консорціум на чолі з Bellcore Labs, Сіракузьким університетом і Co-Operating Systems запустили проект, відомий як FAFNER (Factoring via Network-Enabled Recursion –

рекурсивна факторизація з підтримкою мережевої роботи), що розкладає на множники через Інтернет.

FAFNER був розроблений для факторизації RSA130, використовуючи нову числову методику факторизації, названу NFS (Number Field Sieve), що використовує обчислювальні web-сервери. Учасники проекту тоді використовували CGI (Common Gateway Interface – стандартний інтерфейс обміну даними) для доступу до сервісів підтримки. Ці сервіси включали: поширення програмного забезпечення NFS, проектну документацію, анонімну користувальницьку реєстрацію, поширення завдань і звіти про стан рішення в реальному часі. Скрипти CGI організували підтримку кластерного керування, направляючи індивідуальні робочі станції на рішення задачі під час їхнього простою, щоб мінімізувати завантаженість машин. Таким чином, вони були клієнтами мережі, які використовували протокол HTTP, щоб одержати значення від сервера та відправити результати назад до CGI скрипту сервера.

Щоб зробити цей підхід успішним, були поєднані три фактори:

– реалізація NFS, що дозволила навіть робочим станціям з 4 мегабайтами пам'яті виконувати корисну роботу;

– FAFNER, що підтримував анонімну реєстрацію. Користувачі могли запропонувати свої апаратні ресурси для рішення завдання, не розкриваючи свої дані нікому, крім локального адміністратора сервера;

– консорціум вузлів, який був задіяний, щоб виконати пакет скрипта CGI на локальних машинах, формуючи ієрархічну мережу Web-серверів RSA130, які зменшили потенційне слабке місце адміністрування та дозволили вирішувати завдання цілодобово та автоматично з мінімальним людським втручанням.

Проект FAFNER здобув винагороду в 1995 році на конференції з суперкомп’ютерів Supercomputing’95 у Сан-Дієго. Це проклало шлях до хвилі нових мережевих метаобчислювальних проектів.

1.3.2. I-WAY

Проект I-WAY (The Information Wide Area Year) був експериментальною високоефективною мережею, що об’єднував багато потужних комп'ютерів, і працював над поліпшенням середовищ візуалізації. Проект I-WAY був задуманий на початку 1995 року з ідеєю не побудувати мережу, а інтегрувати існуючі мережі високої пропускної здатності. Використовувалися віртуальні середовища, набори даних і комп'ютери, які розташовувалися на сімнадцяти різних американських сайтах і були підключені за допомогою десятка мереж із змінними смугами пропускання та протоколами, використовуючи різну маршрутизацію та перемикання технологій.

Мережа була заснована на технології асинхронної передачі даних, що у той час тільки розвивалася, а пізніше стала стандартом.. Щоб допомогти стандартизувати інтерфейс програмного забезпечення та керування I-WAY, ключові сайти встановлювали місцеві сервери (I-POP), щоб задіяти їх як шлюзи до I-WAY. Сервери I-POP були робочими станціями UNIX, однаково конфігурованими та мали стандартне програмне середовище за назвою I-Soft.

I-Soft спробував перебороти проблеми різнорідності, масштабованості, продуктивності та безпеки. Кожний сайт, що брав участь в I-WAY, запускав сервер I-POP. Механізми сервера I-POP забезпечили однакову аутентифікацію I-WAY, резервування ресурсу, створення процесу та функції комунікації. Кожний сервер I-POP був доступний через Інтернет і працював у межах системи мережевого захисту свого сайту. У нього також був інтерфейс ATM, що дозволив контролювати потенційне керування ATM-перемикачем сайту.

Проектом I-WAY був розроблений планувальник ресурсів CRB (Computational Resource Broker). CRB складався із протоколів "користувач – CRB" та "CRB – локальний планувальник". Фактична реалізація CRB була структурована в термінах окремого центрального планувальника та множинних локальних демонів планувальника – по одному на кожний сервер I-POP. Центральний планувальник містив черги завдань і таблиць, які у свою чергу розподіляли завдання, містили стан локальних машин та інформацію про стан на розподіленій файловій системі AFS (Andrew File System), яка дозволяє хост-компьютерам, що співробітничають, спільно використати ресурси як через локальну область, так і через глобальні мережі.

В I-POP безпека забезпечувалася використанням telnet-клієнта, зміненого для використання аутентифікації та шифрування (Kerberos authentication and encryption). Крім того, планувальник CRB діяв як аутентифікаційний проксі-сервер, виконуючи наступну аутентифікацію до користувальницьких ресурсів I-WAY. Що стосується файлових систем, I-WAY використав AFS для забезпечення розділеного репозиторію для програмного забезпечення та планувальника інформації. Комірка AFS була встановлена доступною тільки з I-POP.

Для підтримки інструментальних засобів користувальницького рівня та бібліотеки зв'язку нижнього рівня для виконання в середовищі I-WAY був адаптований зв'язок Nexus [9]. Nexus підтримував автоматичні механізми конфігурації, які забезпечили його вибір відповідної конфігурації залежно від використовуваної технології, наприклад, зв'язок через TCP/IP або AAL5 (рівень ATM адаптації для створеного трафіку), використовуючи Інтернет або асинхронну передачу даних. Для використання Nexus також була розширена бібліотека MPICH (переносна реалізація стандарту передачі MPI повідомлень) і CAVEcomm (робота з мережами для системи віртуальної реальності CAVE).

Проект I-WAY визначив кілька типів додатків:

- суперобчислення;
- доступ до віддалених ресурсів;
- віртуальна реальність;
- відео, мережа, GII-Windows.

Проект I-WAY був також успішно продемонстрований на SC 95 у Сан-Дієго. Що є ще більш важливим, досвід роботи та програмне забезпечення, що розроблялося, були використані в подальшому: FAFNER став попередником SETI@home і Distributed.Net, а I-WAY – попередником Globus і Legion.

1.4. ЕВОЛЮЦІЯ GRID: ДРУГЕ ПОКОЛІННЯ (1997 - 2003 РОКИ)

Подальший розвиток і узагальнення ідей метакомп'ютинга на більш ширше коло обчислювальних ресурсів і завдань/додатків вимагав невідкладного рішення широкого кола проблем, пов'язаних з передачею даних, забезпеченням безпеки, управлінням завданнями, доступом до даним, пошуком ресурсів, доступом до них та інше. Підходи до побудови такої Grid-система були описані в [13, 70], і її можна розглядати як представника Grid-систем другого покоління, яким притаманні три основних властивості:

– Різнорідність: Grid використовує різноманітні ресурси, які є різнорідними за походженням, і могли б охопити численні адміністративні домени через потенційно глобальний простір;

– Масштабованість: Grid міг би зростати від декількох ресурсів до тисяч. Це підіймає проблему потенційного зниження рівня продуктивності по мірі росту та поширення Grid. Отже, повинні бути розроблені додатки, які спроможні використовувати наявні ресурси, а також значно покращені засоби аутентифікації користувачів;

– Адаптивність: в Grid відмова ресурсу є скоріше правилом, ніж виключенням. Фактично, з дуже багатьма ресурсами в Grid ймовірність деякого збою ресурсу досить висока. Менеджери ресурсу або додатки повинні динамічно адаптуватися таким чином, щоб вони могли витягти максимальну продуктивність із доступних ресурсів і сервісів.

Як відмічалося, ПГЗ (або програмне забезпечення проміжного шару) використовується для того, щоб приховати різнорідну природу Grid та створити користувачам і додаткам однорідне середовище, забезпечуючи ряд стандартизованих інтерфейсів і безліч сервісів. ПГЗ перебуває між операційною системою і додатками, забезпечуючи додатки безліч сервісами, необхідними для їх коректного функціонування у розподілених гетерогенних середовищах.

Grid – інфраструктура може складатися з будь-яких видів мережевих ресурсів, починаючи від комп'ютерів (обчислювальних ресурсів) до сховищ даних і спеціальних наукових приладів. Традиційно Grid-додатки потребують величезні за масштабом об’єми даних і обчислень. Виникла потреба в створенні нового типу програмного забезпечення Grid систем, яке на відміну від традиційних клієнт-серверних систем, де ресурси управляються і облік користувачів ведеться централізовано в одному довірчому домені, забезпечує:

– динамічну безліч довірчих доменів, що використовують різні механізми безпеки;

– динамічну безліч ресурсів, керованих автономно в рамках одного з доменів;

– динамічну безліч користувачів, що мають облікові записи в деяких доменах;

– доступ в процесі обчислень до відразу декільком ресурсам з різних доменів;

– запуск процесів на декількох ресурсах;

– організацію в процесі обчислень взаємодії між ресурсами з різних доменів;

– взаємодію між запущеними процесами.

Друге покоління базового програмного забезпечення для Grid у своєму розвитку перейшло від ранніх систем типу Globus-GT1[15] і Legion [71], спеціалізованих для конкретних потреб великих і високопродуктивних додатків, до більш універсальних і відкритих середовищ, таких, як Globus-GT3 і Avaki. Разом з базовим програмним забезпеченням у другому поколінні також був розроблений ряд супровідних інструментальних засобів і утиліт для сервісів верхнього рівня, які охопили планувальників ресурсів і брокерів, а також предметно-орієнтовані інтерфейси користувачів і портали. Протягом цього періоду з'явилася й техніка однорангових систем.

До здобутків другого покоління Grid систем слід віднести таке:

– розпочато виконання проекту Globus, направленого на формалізацію і розробку набору (toolkit) базових сервісів для:

– управління ресурсами;

– управління обчисленнями;

– управління даним;

– забезпечення безпеки;

– розроблення і впровадження загальної Grid інфраструктури у вигляді базових сервісів, не залежних від ресурсів і додатків (аутентифікація, авторизація, пошук і розподіл ресурсів, повідомлення про події, облік використання ресурсів, видалений доступ до даним, виявлення відмов і т.д.);

– показ можливості об'єднання базових сервісів в складніші, високорівневі сервіси;

– розроблення ряду програмних компонентів і утиліт, які здійснюють запуск завдань на видалених ресурсах (суперкомп'ютерах, кластерах) через вже існуючі системи управління ресурсами (зазвичай системи пакетної обробки), контроль стану виконання завдань і управління завданнями, збір і доступ до різноманітної інформації про систему і її компоненти тощо, і з допомогою яких можна надати високорівневі послуги користувачам, додаткам, планувальникам ресурсів, брокерам;

– початок низки проектів типу Cactus, WebFlow, DataGrid [72] зі створення великомасштабних обчислювальних та інформаційних мереж ресурсів Grid для аналізу даних.

1.4.1. Основні технології другого покоління

Розглянемо деякі з найбільш значних проектів.цього періоду, пов'язаних з розробкою Grid- інфраструктур, програмним забезпеченням проміжного шару, ключових сервісів, порталів і специфічних додатків.

1.4.1.1. Globus

ПГЗ Globus створює програмну інфраструктуру, що дозволяє додаткам використовувати розподілені різнорідні обчислювальні ресурси як одиничний

віртуальний комп’ютер. Проект Globus, розпочатий ще в1996 році, пов’язаний з розробленням обчислювальних архітектур і виконується в рамках Globus Alliance, куди увійшли Argonne National Laboratory, University of Chicago;EPCC, University of Edinburgh; National Center for Supercomputing Applications (NCSA);Northern Illinois University, High Performance Computing Laboratory; Royal Institute of Technology, Sweden;Univa Corporation; University of Southern California Information Sciences Institute. Згідно проекту, обчислювальний Grid є апаратною та програмною інфраструктурою, що забезпечує надійний, однозначний доступ до високопродуктивних обчислювальних можливостей, незважаючи на географічний розподіл як ресурсів, так і користувачів. Центральний елемент системи Globus – Globus Toolkit (інструментарій), що визначає основні сервіси та можливості, необхідні для створення обчислювального Grid середовища. Інструментарій складається з ряду компонентів, які забезпечують основні сервіси, такі як безпека, місце розташування ресурсів, керування ресурсами, зв'язок.

Для обчислювальних Grid є необхідною підтримка широкої різноманітності моделей програмування та створення додатків. Отже, замість того, щоб забезпечити однорідну модель програмування, таку як, наприклад, об’єктно-орієнтована модель, Globus Toolkit забезпечує безліч сервісів зі специфічними властивостями, які можуть підтримувати різні засоби програмування додатків. Ця методологія можлива тільки тоді, коли сервіси є різними та мають чіткі інтерфейси (API), які можуть бути включені в додатки або інструментальні засоби програмування шляхом нарощування.

Globus створений як багаторівнева архітектура, у якій глобальні сервіси високого рівня побудовані поверх основних локальних сервісів нижнього рівня. Globus Toolkit є модульним, і додаток може експлуатувати особливості Globus, такі як керування ресурсами або інформаційною інфраструктурою, не використовуючи бібліотеки комунікації Globus. Globus Toolkit сьогодні складається з наступних елементів (точний набір залежить від версії Globus):

– GRAM («Globus Toolkit Resource Allocation Manager» – менеджер розподілу ресурсів Globus Toolkit), заснований на HTTP, який використовується для розподілу обчислювальних ресурсів і для контролю обчислення на цих ресурсах;

– розширена версія протоколу передачі файлів GridFTP (Grid File Transfer Protocol), шо використовується для доступу до даних. Розширення включають використання протоколів забезпечення безпеки з’єднання, доступу до файлу і керування паралелізмом для високошвидкісних передач;

– аутентифікація та зв'язані з нею функції захисту (GSI – Grid Security Infrastructure, інфраструктура безпеки Grid);

– розподілений доступ до структури та інформації про стан структури, що заснована на LDAP (Lightweight Directory Access Protocol – «протокол легкого доступу до директорій»). Цей сервіс використовується для визначення стандартного інформаційного протоколу ресурсу та пов'язаної з ним інформаційної моделі;

– доступ до даних через послідовні та паралельні інтерфейси (GASS – Global Access to Secondary Storage, «глобальний доступ до зовнішньої пам’яті»), включаючи інтерфейс, заснований на GridFTP;

– конструкція, кешування та місцерозташування виконуваних програм (GEM – Globus Executable Management, «керування виконуваною програмою Globus»);

– резервування ресурсів і розподіл ресурсів (GARA – Globus Advanced Reservation and Allocation).

Globus розвивався від свого прототипу в першому поколінні – проекту I-WAY, через версію 1 (GT1, 1998 рік) до версії 3 (GT3, 2003 рік). Протоколи та сервіси, які надав Globus, досить динамічно змінювалися протягом свого розвитку. Акцент Globus перейшов від підтримки одних тільки високоефективних додатків до підтримки більше розповсюджених сервісів, які можуть підтримувати віртуальні організації. Еволюція Globus продовжується введенням архітектури OGSA (Open Grid Service Architecture, відкрита архітектура сервісів Grid) [18,73], і архітектури Grid, заснованої на Web-сервісах.

1.4.1.2. Legion

ПГЗ Legion є «метасистемою», заснованою на об'єктах, і розробленою у Вірджинському університеті. Legion сформував програмну інфраструктуру таким чином, щоб система різнорідних, географічно розповсюджених і високоефективних машин могла вільно взаємодіяти. Legion зробив спробу надати користувачам на їхніх робочих станціях окрему інтегровану інфраструктуру незалежно від масштабу, фізичного місця розташування, мови програмування та основної операційної системи користувача.

Legion відрізнявся від Globus у своєму підході до забезпечення середовища Grid: він формував всі свої компоненти у вигляді об'єктів. У цієї методології є всі звичайні переваги об’єктно-орієнтованого підходу, такі як абстракція даних, інкапсуляція, спадкування та поліморфізм.

Legion визначив API до ряду основних об'єктів, які підтримують основні сервіси, необхідні цій метасистемі. У системи Legion був наступний набір основних типів об'єктів:

– класи та метакласи (класи можна вважати менеджерами та організаторами, метакласи – це класи класів);

– об'єкти Host (абстракції обробки ресурсів, вони можуть представити одиничний процесор або безліч хост-компьютерів і процесорів);

– об'єкти Vault (являють собою постійну пам'ять, але тільки з метою підтримки постійного стану об'єктного завдання);

– об'єкти реалізації (Implementation) і кеши (Caches). Об'єкти реалізації приховують подробиці реалізацій об'єктів пам'яті та можуть вважатися як еквівалентні виконуваним програмам в UNIX;

– агенти закріплення (Binding Agents) відображають об'єктні ідентифікатори фізичним адресатам;

– об'єкти Context і простори Context відображають контекстні імена ідентифікаторів об'єктів Legion, дозволяючи користувачам надавати об'єктам імена, що є рядками довільної довжини.

Legion був вперше випущений у листопаді 1997 року. З тих пір компоненти, які становлять Legion, продовжили розвиватися. У серпні 1998 року компанія Applied Metacomputing одержала дозвіл на комерційне використання Legion. У червні 2001 року естафета була перехоплена корпорацією Avaki (Avaki Corporation).

1.4.1.3. Розподілені об'єктні системи

CORBA (The Common Object Request Broker Architecture) є відкритою розподіленою об'єктно-обчислювальною інфраструктурою, стандартизованою OMG (Object Management Group, група по керуванню об'єктами). CORBA автоматизує функціонування загальних мереж, програмуючи такі завдання, як об'єктна реєстрація, місце розташування та активація; демультиплексування запиту; синхронізація кадрів і обробка помилок; сортування та десортування параметра, диспетчеризація операції. Хоча CORBA забезпечує широкий набір сервісів, проте вона не містить сервіси розподілу та планування рівня Grid, які є в Globus, однак CORBA цілком здатна об’єднатися з Grid. CORBA є середовищем, що не залежить від мови та близько асоціюється з UML (Unified Modeling Language). Однин із недоліків CORBA – це те, що вона більше підходить для внутрішніх мереж, а не для поширення через Інтернет, і тому існують труднощі в організації, наприклад, виконання операцій через системи мережевого захисту. Крім того, не забезпечується організація взаємодії в реальному часі та підтримка мультимедіа.

1.4.1.4. Java

Java створює окрему структуру для реалізації розподілених об'єктних систем. Певною мірою віртуальна машина Java (JVM, Java Virtual Machine) з Java-додатками та сервісами вирішує проблеми, пов'язані з різнорідними системами, організовуючи переносні програми та розподілену об'єктну модель через протокол RMI. Однак, використання Java саме по собі містить певні недоліки, і головним з них є низька обчислювальна швидкість. Java Grande («Додаток Grande») є будь-яким додатком, науковим або індустріальним, котрий вимагає великої кількості обчислювальних ресурсів, у тому числі і віддалених, для рішення однієї або більше задач. Мова Java була також обрана для створення альтернативного програмного забезпечення проміжного шару UNICORE (UNiform Interface to COmputer REsources) в 2003 році. Головні компоненти UNICORE: агент підготовки завдань (JPA); контролер монітора завдань (JMC); сервер https UNICORE, так званий шлюз (Gateway); супервізор мережних завдань (NJS); графічний інтерфейс користувача, заснований на Java-аплетах, з інтерактивною довідкою й засобами допомоги.

Втрати в обчислювальній швидкості виконання Java додатків можуть бути компенсовані значним прискоренням строків їх розробки, тим самим представляючи більше широкий погляд на підходи до розроблення додатків Grid.

1.4.1.5. Jini і протокол RMI

Технологія Jini була розроблена для того, щоб сформувати програмну інфраструктуру, яка надає можливість створити середовище розподілених обчислень. В Jini додатки можуть бути написані мовою Java і оброблятися з використанням механізму Java RMI (Java Remote Method Invocation). Навіть при тому, що Jini написана на чистому Java, ні від клієнтів Jini, ні від сервісів не вимагається бути чистими Java. Вони можуть включати надбудови Java навколо коду на іншій мові, або навіть бути повністю написані на деякій іншій мові. Це дозволяє Jini використовуватися поза стандартною структурою Java і зв'язувати сервіс і клієнтів від безлічі джерел.

Якщо копати глибше, технологія Jini, насамперед, зацікавлена у зв'язку між пристроями, а не у виконанні цими пристроями яких-небудь завдань. Фактична реалізація сервісу може бути здійснена апаратними засобами і/або програмним забезпеченням. Сервіси в технології Jini- є взаємно обізнаними, і їх множину (родина) взагалі вважають розміром робочої групи. Сервіс пошуку і LUS (LookUp Service) може бути експортований в інші співтовариства, у такий спосіб забезпечуючи взаємодію між декількома ізольованими співтовариствами.

В Jini новий пристрій або програмний сервіс можуть бути підключені до мережі і там оголосити про свою присутність. Клієнти, які бажають використати такий сервіс, можуть тоді визначити його місцезнаходження та дати йому своє ім'я, що відразу дозволяє почати виконувати завдання. Jini заснований на протоколі RMI, що вводить деякі обмеження. Крім того, Jini не є розподіленою операційною системою, оскільки операційна система надає такі послуги, як доступ до файлу, планування процесора та користувальницькі входи в систему. Ось п'ять ключових концепцій Jini:

– Lookup (пошук) – пошук сервісу та завантаження коду, що необхідний для звертання до нього;

– Discovery (виявлення) – швидкий(миттєвий) пошук родини або об'єднання;

– Leasing (лізинг) – обмежений у часі доступ до сервісу;

– Remote Events (вилучені події) – сервіс А подає повідомлення сервісу В про зміну стану сервісу А. Lookup здатний зареєструвати всі сервіси нового сервісу;

– Transactions (трансакції) – використовуються для гарантії того, що розподілений стан системи залишається несуперечливим і однозначним.

1.4.1.6. The Common Component Architecture Forum

The Common Component Architecture Forum (форум архітектури загальних компонентів) [73] ставить за мету визначити мінімальний набір стандартних властивостей, які високопродуктивна структура компонентів повинна була б забезпечити. Як і CORBA, він підтримує програмування компонентів, але воно відрізняється від інших підходів програмування компонентів, наголошуючи на підтримку абстракцій, необхідних для високоефективного програмування. Для здійснення сервісів у межах складової

структури можуть використовуватися основні технології, описані в попередній главі, – Globus або Legion. Ідея використати структури компонентів для складної розробки міждисциплінарних додатків стає все більш та більш популярною. Такі системи дозволяють програмістам прискорити проектну розробку, вводячи абстракції більш високого рівня та дозволяючи повторне використання коду. Вони також забезпечують певні складові інтерфейсів, які полегшують завдання взаємодії групи: такий стандарт поліпшить взаємну функціональну сумісність компонентів, розроблених різними групами на різних установах. Ці потенційні вигоди надихнули існуючі групи дослідників в багатьох лабораторіях і університетах на розвиток і проведення експериментів з такими системами. Існує потреба в стандартах функціональної сумісності, щоб уникнути фрагментації.

1.4.1.7. Посередники та планувальники ресурсів Grid

Існує кілька доступних систем, які фокусуються на здійсненні групування та планування ресурсів. Потрібно відзначити, що всі пакети, що тут перераховані, виникли як системи керування завданнями та локальними платформами розподілених обчислень:

– Condor [74] – пакет програм для виконання пакетних завдань на безлічі UNIX платформах, особливо на тих, які в конкретний час простоюють. Основні особливості Condor – автоматичне місцерозташування ресурсу та розподіл завдання, перевірка звернень, міграція процесів (рис.1.3). Ці особливості здійснені без модифікації основного ядра UNIX. Однак, для користувача необхідно зв'язати вихідний текст із бібліотеками Condor. Condor контролює діяльність всіх розподілених обчислювальних ресурсів, і ті машини, які є доступними, поміщені в кластер. Потім машини розподіляються для виконання завдань. Такий кластер є динамічним об'єктом, туди поміщуються робочі станції, коли вони починають простоювати, і видаляються звідти, як тільки починають виконувати роботу.

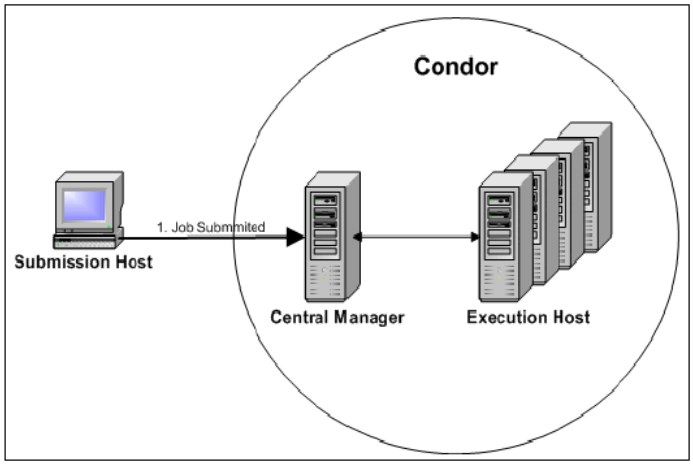


Рисунок 1.3. – Процедура розполілу завдань в Condor

– PBS (The Portable Batch System, портативна система групування) є системою керування організації черги та робочого навантаження партії

(спочатку розроблена для NASA). Вона працює на безлічі платформ UNIX, від кластерів до суперкомп'ютерів. Планувальник завдання PBS дозволяє вузлам установлювати свою власну політику планування для виконання завдання в часі та у просторі. PBS пристосована до широкої різноманітності адміністративної політики і забезпечує розширювану аутентифікацію та модель захисту. PBS забезпечує також графічний інтерфейс користувача для подання задачі, трекінга та адміністративних цілей.

– SGE (The Sun Grid Engine) заснований на програмному забезпеченні, розробленому Genias, відомим як Codine/GRM. Користувач представляє SGE завдання і повідомляє необхідні вимоги для цього завдання. Завдання надходить до черзі, що розташована на серверах. SGE оцінює завдання і потім знаходить для виконання серед завдань в черзі завдання з найвищим пріоритетом або найдовшим часом очікування. У такий спосіб передаються нові завдання до найбільш необхідної або найменш навантаженої черзі.

– LSF (The Load Sharing Facility, розділений засіб завантаження) – комерційна система від корпорації Platform Computing Corp. Система LSF, що розвилася від системи Utopia, розробленої в університеті міста Торонто, і в наш час є найбільш широко використовуваною комерційною системою керування завданнями. LSF включає розподілені процедури чергового завантаження та групування програмного забезпечення, яке управляє, контролює та аналізує ресурси в мережі різнорідних комп'ютерів, а також відзначається стійкістю до відмови системи.

– SRB (Storage Resource Broker, брокер ресурсу пам'яті) був розроблений у суперкомп'ютерному центрі Сан-Дієго (San Diego Supercomputer Centre, SDSC), щоб забезпечити «однорідний доступ до розподіленої пам'яті» через чіткий API. SRB підтримує дублювання файлу, і це може відбутися як у режимі автономної роботи, так і прямому режимі. Взаємодія з SRB здійснюється через графічний користувальницький інтерфейс. Сервери SRB можуть бути об'єднаними. SRB управляє адміністратор, з можливістю створювати групи користувачів. Головною особливістю SRB є те, що він підтримує метадані, що пов'язані з розподіленою файловою системою, такі як місцерозташування, розмір і інформація про дату створення. Він також підтримує таке поняття як метадані прикладного шару, певні для інформаційного наповнення, яке не може бути узагальнене через всі набори даних. На відміну від традиційних мережевих файлових систем, SRB привабливий для додатків Grid, у яких він має справу з більшими томами даних, що можуть переповнити індивідуальні запам'ятовувальні пристрої, тому що він має справу з метаданими та використовує дублювання файлів.

– Nimrod-G є брокером ресурсів Grid, що виконує керування ресурсами та планування маршрутів і додатків, що обробляють задачі. Він складається із чотирьох компонентів: механізм, що обробляє завдання; планувальник; диспетчер; агенти ресурсів. Механізм обробки завдань Nimrod-G дозволяє планувальникам, які визначаються користувачем, налаштованим додаткам або середовищам рішення завдань (наприклад, ActiveSheets) бути

«включеними», замість заданих за замовчуванням компонентів. Диспетчер використає Globus для того, щоб розгорнути Nimrod-G агенти на віддалених ресурсах для управління виконанням призначених завдань. Планувальник Nimrod-G має здатність орендувати ресурси Grid і сервіси залежно від їхньої продуктивності, витрат і доцільності. Планувальник підтримує відкриття ресурсу, вибір, планування і виконання користувальницьких завдань на віддалених ресурсах. Користувачі можуть установити крайній термін, до якого необхідно одержати результати, і брокер Nimrod-G шукає найбільш підходящі ресурси, доступні в Grid, і використовує їх для того, щоб виконати роботу в строк і спробувати мінімізувати вартість виконання завдання. Nimrod-G підтримує визначення користувачем крайнього строку та обмеження його бюджету, для того, щоб оптимізувати і управляти попитом та пропозицією ресурсів в Grid, використовуючи безліч сервісів та ресурсів.

– Grid-Портали, які дозволяють вченим і дослідникам звертатися до ресурсів через Web-інтерфейс.На відміну від типових порталів Інтернет, Grid-портал може також забезпечити доступ до ресурсів Grid. Наприклад, Grid-портал може підтвердити дійсність користувачів, дозволити їм звертатися до віддалених ресурсів, допомагати їм ухвалювати рішення щодо планування завдань і дозволяти користувачам звертатися та управляти інформацією ресурсу, отриманої та збереженої на віддаленій базі даних. Доступ до Grid-порталу може також бути індивідуалізований за допомогою конфігурацій, які створені та збережені для кожного користувача порталу. Ці, а також інші атрибути, роблять Grid-портали відповідними засобами для користувачів додатка Grid для звертання до ресурсів Grid. Прикладом може служити портал HotPage розроблений для одноразового входження в Grid – систему і спрощення доступу до її розподілених ресурсів.

Як тільки виникло друге покоління компонентів Grid, багато міжнародних груп запустили проекти, які інтегрували ці компоненти в свої системи ПГЗ. Наприклад, таким чином в 2003 році було створено програмне забезпечення проміжного шару NorduGrid для обслуговування потреб країн північної Європи і розпочата розробка програмного забезпечення проміжного шару gLite для підтримки проектів CERN.

1.4.2. Однорангові обчислення

Одним з підходів до рішення проблем масштабування є децентралізація. Традиційна клієнт-серверна модель може стати вузьким місцем у підвищенні продуктивності і єдиним джерелом відмов, хоча усе ще залишається основною, тому що децентралізація несе із собою власні проблеми. Проте однорангові (Peer-to-Peer – P2P) обчислення [2] і Інтернет-обчислення (SETI@home і Entropia [75]) є прикладами досить загальних обчислювальних структур, які користуються перевагою глобально розподілених ресурсів.

В обчисленнях P2P комп'ютери розділяють дані й ресурси, такі як процесорний час й ємність довгострокової пам'яті, через і Інтернет або приватні мережі. Комп'ютери можуть також обмінюватися даними безпосередньо й управляти обчислювальними завданнями без використання центральних

серверів. Це дозволяє масштабувати обчислення P2P більш ефективно, ніж традиційні клієнт-серверні системи, які повинні розгорнути для цього серверну інфраструктуру для забезпечення можливостей розширення. Децентралізація клієнта й сервера одночасно є привабливою не тільки щодо масштабованості, але й відмовостійкості. Однак є деякі перешкоди на шляху масового прийняття систем P2P, які зводяться до наступного:

– від персональних комп'ютерів і робочих станцій, використовуваних у складних додатках P2P, будуть потрібні більші обчислювальні можливості для виконання додаткового навантаження по зв'язку й захисту інформації, які звичайно несуть сервери;

– захист є серйозною проблемою, оскільки комп'ютери для додатків P2P вимагають доступу до ресурсів інших комп'ютерів (пам'яті, жорстким дискам і т.д.). Завантаження файлів з інших комп'ютерів робить системи уразливими з боку вірусів;

– системи P2P повинні працювати з різнорідними ресурсами, що використовуютьють різні компоненти роботи з мережами й операційними системами;

– однією із найбільших проблем P2P-обчислень є можливість пристроїв знаходити один одного в обчислювальній структурі, у якій відсутнє центральне керування.

Ідеї P2P починають приваблювати все більше дослідників. В 2001 році Sun Microsystems оповістила про початок проекту JXTA [76] з відкритим кодом для інфраструктури й додатків P2P.

1.5. Еволюція GRID: третє покоління для е-науки (з 2004 року)

Однак друге покоління ПГЗ не забезпечило повну функціональну сумісність створеного Grid програмного забезпечення, що є важливою умовою реалізації великомасштабних обчислень. Оскільки подальші рішення Grid продовжували досліджуватися, стали очевидними інші аспекти розробки Grid. Щоб побудувати нові Grid – додатки, бажано багаторазово використати існуючі компоненти та інформаційні ресурси, і гнучко ними оперувати.

Дві ключових особливості Grid.систем третього покоління – наголос на прийняття сервісно-орієнтованої моделі та зростаюча увага до метаданих. Фактично сервісно-орієнтований підхід припускає, що гнучке застосування ресурсів Grid у додатках Grid вимагатиме інформацію про функціональні можливості, характеристики та інтерфейси різних компонентів, і ця інформація повинна бути погодженою, щоб можна було її обробляти комп’ютером.

Були введені нові терміни «розподілене співробітництво» і «віртуальна організація». В третьому поколінні сформувалося більш цілісне уявлення про виконання Grid обчислень, і можна сказати, що справа йдеться скоріше про інфраструктуру для е-Науки, ніж про поліпшення уже розробленої існуючої технології. Очікуване використання обчислювальних засобів з масовим паралелізмом – тільки частина картини, що намалювалася, існує також величезна кількість користувачів, отже розподілені обчислення не були прерогативою тільки комп’ютерів з масовим паралелізмом.

Е-Наука (e-Science) – порівняно новий термін, який придбав особливу популярність після запуску головного британського проекту e-Science [46]. Він охоплює новий підхід до науки, що включає розподілену глобальну співпрацю, яка забезпечуються за допомогою Інтернету і використання величезних масивів даних, комп'ютерних ресурсів терамасштабу і високопродуктивної візуалізації. Сутність e-Science проілюстрована на рис.1.4. Останнє десятиліття фокусується на інтеграції науки і інженерії з обчислювальною технікою. За останні 50 років наукова практика продемонструвала зростаючу силу спілкування і важливість колективної співпраці в наукових відкриттях. Раніше вчені спілкувалися за допомогою кораблів і поштових голубів. Тепер вони спілкуються за допомогою авіапошти, телефону, електронної пошти та Інтернету. Співпраця може бути і «реальною» завдяки електронним засобам, тому можна сказати, що Grid – це інфраструктура, що забезпечує існування співробітницької науки.

Grid може забезпечити підтримку віддаленого спілкування вчених в реальному часі. Особливо важливою є інфраструктура для підтримки розподілених ресурсів – тут є багато ключових сервісів: безпека, масштабування і управління, реєстрація і пошук, та інтерфейси Web-сервісів, орієнтовані на повідомлення, для забезпечення могутніх механізмів співробітництва. Всі головні Grid-сервіси і інфраструктура забезпечують співпрацю і є вкрай значущими для суспільства.

Рисунок 1.4. – Використання інформаційних технологій в e-Science

Існує безліч прикладів, які ілюструють вражаюче зростання кількості наукових даних, що генеруються. Як приклад можна розглянути завдання контролю робочого стану промислового устаткування. Британська e-Science програма профінансувала проект DAME (Distributed Aircraft Maintenance Environment) – розподілене оточення технічного обслуговування авіації) [67], який присвячений аналізу тих даних, що генеруються аеродвигунами компанії Rolls Royce. Вважається, що зараз використовується багато тисяч двигунів

виробництва Rolls Royce. Для прикладу, кожен трансатлантичний переліт здійснений двигуном, генерує біля гігабайта даних – від датчиків тиску, температури і коливання. Мета проекту – передати невелику частинку цих первинних даних для аналізу і порівняння з даними, які зберігаються в трьох центрах в різних точках миру. Шляхом отримання перших результатів Rolls Royce сподівається збільшити період між плановими техобслуговуваннями, таким чином збільшивши рентабельність двигунів. Датчики двигунів генеруватимуть безліч петабайт даних за рік але в реальному часі повинні будуть ухвалюватися рішення про те скільки даних аналізувати, скільки передавати для подальшого аналізу і скільки архівувати.

Подібні, а то і більші, об’єми даних генеруватимуться іншими експериментами з високопродуктивними сенсорами в таких сферах як спостереження за Землею і навколишнім середовищем, фізика високих енергій і, звичайно, контроль людського здоров'я [14].

З цих прикладів очевидно, що e-Science дані, які генеруються сенсорами, супутниками, складними комп'ютерними моделюваннями, високопродуктивними пристроями, науковими фотографіями і інше скоро перевищать дані, зібрані за всю історію наукових досліджень. На сьогоднішній день, розмір найбільших комерційних баз даних варіюється від десятків до сотень терабайт. Очікується, що в найближчі роки ситуація різко зміниться, і об’єм наукових даних значно перевищить об‘єми комерційних систем. Цей переломний момент неминуче приведе як до нових труднощів, так і до нових можливостей. Саме з цієї причини, можливості наступного покоління Grid програмного забезпечення по доступу до даних, інтеграції даних і об’єднанню даних зіграють ключову роль як для e-Science так і для e-Business.

Об'єми даних настільки значні, що людині просто не під силу проаналізувати їх самостійно, хоча необхідність проведення такого аналізу цілком очевидна, адже в цих "сирих даних" укладені знання, які можуть бути використаний при ухваленні рішень. Для того, щоб провести автоматичний аналіз даних, використовується Data Mining (добування, «витягання» знань) [77]. Це нова технологія інтелектуального аналізу даних з метою виявлення прихованих закономірностей у вигляді значущих особливостей, кореляцій, тенденцій і шаблонів. Сучасні системи добування даних використовують засновані на методах штучного інтелекту засоби уявлення і інтерпретації, що і дозволяє знаходити розчинену в терабайтних сховищах не очевидну, але вельми цінну інформацію. Фактично, ми говоримо про те, що в процесі Data mining система не відштовхується від наперед висунутих гіпотез, а пропонує їх сама на основі аналізу.

В основу сучасної технології Data Mining встановлена концепція шаблонів (pattern), що відображають фрагменти багатоаспектних взаємозв‘язків в даних. Цими шаблонами є закономірності, властиві підвибіркам даних, які можуть бути компактно виражені у формі, зрозумілій людині. Пошук шаблонів проводиться методами, не обмеженими рамками апріорних припущень про структуру вибірки і вид розподілів значень аналізованих показників.

Висунута IBM концепція системи автономних обчислень [23] містить наступні властивості Grid систем третього покоління:

– необхідність деталізованої інформації про компоненти та їх стан для автоматизації процесів обчислення шляхом створення його сценарію;

– автоматична та динамічна конфігурація та реконфігурація;

– прагнення оптимізувати свою поведінку для досягнення поставлених цілей;

– здатність відновлення після збою;

– самозахист від вірусної атаки;

– знання про власне середовище;

– використання відкритих стандартів;

– забезпечення оптимального використання ресурсів.

Третє покоління Grid систем націлене, насамперед, на створення «загального дослідницького простору» («collaboratory»), визначеного в 1993 році американським Національним Науковим Фондом як «центр без стін, у якому національні дослідники можуть виконувати дослідження незалежно від географічного місця розташування шляхом взаємодії з колегами, спільно використовуючи інструментарії, дані й обчислювальний ресурс і звертаючись за інформацією до цифрових бібліотек». Успішними прикладами таких зусиль можуть бути проекти Grid2003 [47], що призначений поєднати могутні обчислювальні центри США; проект EGEE (Enabling Grids for E-science in Europe) [14], метою якого є об'єднання національних, регіональних і тематичних Grid – систем в єдину цільну Grid -інфраструктуру для підтримки наукових досліджень; проект Access Grid [78], присвячений колекції ресурсів, які підтримують співробітництво людей через Grid, включаючи великомасштабні розподілені зустрічі та навчання.

До послуг користувачів, що співпрацюють в Grid системі, надаються наступні можливості:

– Оперативні дані від експериментального обладнання;

– передача відеоінформації в реальному часі (Web-камери) в індивідуальному режимі або у вигляді розсилання або групової передачі (наприклад, MBONE);

– відеоконференції;

– система групових Інтернет-дискусій;

– миттєві системи обміну повідомленнями;

– багатокористувацькі системи віртуальної реальності,

– Інтернет-чати;

– спільні віртуальні середовища.

Всіх ці елементи відіграють роль у підтримці е-Науки. Зокрема вони підтримують розширення Е-Науки на нові співтовариства, які перетинають поточні організаційні та географічні границі.

Grid системи третього покоління впевнено крокують до Семантичних Grid, заснованих на використанні метаданих і онтологій, у яких інформація розуміється як тільки як дані, що мають значення, але і знання, які

здобуваються [79], використаються, представляються, публікуються й підтримуються, щоб допомогти Е-вченим досягати їхніх специфічних цілей. Знання розуміються як інформація, застосована для досягнення мети, рішення проблеми або ухвалення рішення. Семантичний Grid охоплює всі три концептуальні шари Grid: знання, інформація й обчислення/дані. Ці додаткові шари в остаточному підсумку забезпечать багатий, безшовний і розповсюджений доступ, що поширюється на глобально розподілені гетерогенні ресурси. Одна із стандартних методик, що використовуються в управлінні знаннями, – це розроблення порталу знань. Стандартне визначення для нього таке: портал – це працюючий на базі Web додаток, що забезпечує засоби для накопичення, пристосування та персоналізації даних.

Особлива роль належить інфраструктурі метаданих: дані від експериментального встаткування можуть бути виражені відповідно до онтології, тим самим ці дані можуть бути оброблені програмами в такий же спосіб, як статичні дані. На початку 21 століття ми бачимо величезний прорив у телекомунікаціях, коли зявляються глобальні мережі типу GEANT [7]. Сьогоднішні всюдисущі мобільні телефони та кишенькові комп'ютери – це тільки початок глибшого прориву, націленого на зростаючу легкість надання нам вичерпної інформації про світ, що нас оточує. За наступне десятиліття для розробників додатків все важливіше буде інтегрувати нові пристрої і нові джерела інформації з Grid. Датчики та їх мережі, закладені в мостах, дорогах, одязі тощо, забезпечать величезне джерело даних. Аналіз інформації в реальному часі зіграє ще важливішу роль в охороні здоров'я, безпеці, економічній стабільності тощо. Інтеграція нових пристроїв забезпечить Grid-співтовариству розроблення програмного забезпечення та додатків, але ще й створить абсолютно новий рівень потенціалу для наукових досягнень.

До найбільш вагомих здобутків Grid систем третього покоління, отриманих до тепер, можна віднести:

– злиття Grid-технологій і технологій Web-сервісів, формування Grid сервісу як спеціального розширення Web-сервісу шляхом підтримки екземплярів Grid-сервісу, що мають стан і, можливо, обмежений час життя; З кожним екземпляром Grid-сервісу пов'язані дані сервісу, тобто інформація, структурована у вигляді набору іменованих XML-елементів, що типізуються (service data elements, SDE;

– забезпечення інтероперабельності різних реалізацій сервісів, визначення стандартизованих інтерфейсів сервісів (OGSI), визначення протоколу(-ів) для виклику певного інтерфейсу, домовленість про стандартний набір підтримуваних протоколів, при чому для кожного із стандартних інтерфейсів визначений набір елементів даних сервісу, які повинні підтримуватися будь-яким екземпляром сервісу, що реалізовує даний інтерфейс;

– запропонування спільними зусиллями GT, IBM та інших компаній набору специфікацій під ім'ям WS-Resource Framework (WSRF), який спирається на ту ж архітектуру OGSA, на загальновизнані стандарти Web-

сервісів, при чому зберігається багато елементів OGSI, але використовується інша термінологія і розширюються можливості OGSI;

– створення програмному Grid забезпечення Globus Toolkit GT4 як відкритої реалізації WSRF і засоби розробки клієнтських і серверних додатків на мовах Java, C++ і Python; при цьому GT4 не повністю сумісний з попереднім GT3.

Історія Globus Toolkit наочно демонструє еволюцію перспективної технології від суто академічного проекту з вузьким колом користувачів до загальноприйнятого стандарту, що користується широкою підтримкою IT-індустрії у цілому світі. Компанії HP, IBM, Інтел і Microsoft оголосили у березні 2006 про свій намір співпрацювати разом, щоб "розвивати загальний набір мережевих сервісів для ресурсів, подій і менеджменту, який може широко підтримуватися різними платформами". Вони також склали меморандум про співпрацю і визначили, якими будуть ці специфікації. Цей меморандум пропонує шляхи до узгодження двох подібних, але конкуруючих підходів: Web Services Distributed Management (WSDM), шо включає WSRF і WS-Notification (WS-N), який підтримується IBM, HP, та іншими; з специфікаціями WS-Management, що включвють WS-Transfer, WS-Eventing, WS-Enumeration і підтримуються Microsoft, Інтел, та ін. Виданий меморандум свідчить, що нові специфікації, які повинні бути створені, міститимуть всі основні поняття, введені раніше в Open Grid Services Infrastructure (OGSI) і згодом включені в WSRF/WS-N. Отже ця ініціатива є сприятливою звісткою про те, що Grid організації і Globus Alliance протягом найближчих 5-ти років будуть спільно працювати у напрямку розробки промислових стандартів для системного менеджменту, заснованого на Web сервісах.

Розглянемо детальніше найбільш видатні досягнення цього періоду.

1.5.1. Сервісно-орієнтована архітектура

До 2003 року з'явився ряд архітектур Grid, запропонованих у різних проектах. Наприклад, в роботі [13] запропонувала відома багатошарова модель, а в проекті IBM Information Power Grid [80] – модель з великим набором сервісів, знову ж таки упорядкованих пошарово. До цього часу модель Web-сервісів також придбала популярність, обіцяючи стандарти для підтримки сервісно-орієнтованого підходу. Ще одне з дослідницьких співтовариств розпочало роботу в області обчислень, заснованих на застосуванні програмних агентів: програмні агенти можуть бути визначені як виробники, споживачі та брокери сервісів [81]. В цілому, стало очевидним те, що сервісно-орієнтована парадигма забезпечує гнучкість, необхідну для третього покоління Grid.

Велике значення мало створення консорціумом W3C стандартів Web-сервісів таких як:

– SOAP (протокол XML). SOAP забезпечує пакет, що формує дані XML для передачі через Web-інфраструктуру (наприклад через HTTP, через кеш та проксі-сервери), з викликом віддалених процедур (Remote Procedure Calls, RPCs) і механізмом послідовного впорядкування, заснованого на типах

даних XML Schema. SOAP розробляється консорціумом W3C у співробітництві із Групою інженерної підтримки Internet (Internet Engineering Task Force, IETF).

– Мова опису Web-сервісів (WSDL). Описує сервіс в XML, використовуючи XML Schema; є також відображення на RDF (Resource Description Framework – структура опису ресурсу). Певною мірою WSDL подібний мові опису інтерфейсу IDL (interface definition language).

– UDDI (Universal Description Discovery and Integration, універсальний пошук, опис і інтеграція). Це – специфікація для розподілених регістрів Web-сервісів. UDDI підтримує процедури «опублікувати, знайти та зв'язати»: сервісний провайдер описує та публікує подробиці сервісу в каталозі; запитувач сервісу ставить запити до системного реєстру для того, щоб знайти сервісних провайдерів; сервіси «зв'язуються», використовуючи технічні можливості, надані UDDI. UDDI ґрунтується на XML і SOAP.

– Мова WSFL (Web Services Flow Language, мова потоків даних Web-сервісів) є пропозицією IBM, що визначає робочий процес у вигляді комбінації Web-сервісів; XLANG від Microsoft підтримує складні транзакції та множинні Web-сервіси. Очікується об'єднана пропозиція цих двох стандартів.

– WSMF (Web Services Modelling Framework) забезпечує концептуальну модель для розробки та опису Web-сервісів, засновану на принципах максимального роз'єднання та масштабованому сервісі посередництва.

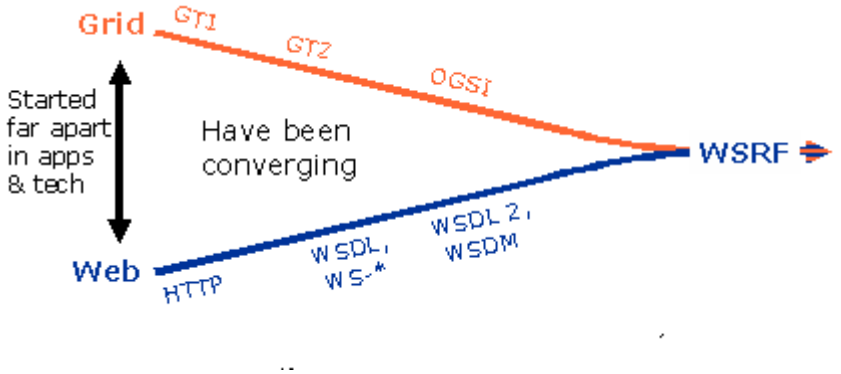


Рисунок 1.5. – Процес об’єднання Web- і Grid сервісів

Web-сервіси близько відповідають вимогам до систем Grid третього покоління: вони підтримують сервісно-орієнтований підхід і підтримають стандарти для полегшення інформаційних процесів, таких як опис сервісу (рис.1.5).

1.5.2. Архітектура OGSA

Структура OGSA (Open Grid Services Architecture, відкрита архітектура Grid-сервісів) – спільне бачення Globus-IBM для об’єднання Web-сервісів і Grid-обчислення, була представлена на Глобальному Grid форумі (GGF) у Торонто ще в лютому 2002 року. OGSA описана в роботах [4,13]. GGF установив робочу групу відкритих Grid-сервісів для того, щоб створити огляд і

вдосконалити архітектуру Grid-сервісів і документів, які формують технічну специфікацію.

OGSA підтримує створення, обслуговування та додаток ансамблів сервісів, які використовуються у віртуальних організаціях (ВО). Тут сервіс визначений як об'єкт із підтримкою мережевої роботи, що забезпечує деякі такі можливості, як обчислювальні ресурси, ресурси пам'яті, мережі, програми та бази даних. Це пристосовує створення Web-сервісів для виконання деяких вимог, специфічних для Grid. Нижче наведені стандартні інтерфейси, наявні в OGSA:

– Виявлення (discovery): клієнти потребують механізми для того, щоб знайти доступні сервіси та визначити особливості цих сервісів і мати можливість їх конфігурувати відповідно своїх запитів до цих сервісів.

– Динамічне створення сервісу (Dynamic service creation): стандартний інтерфейс і семантика, що повинні забезпечити створення будь-якого сервісу.

– Довічне керування (Lifetime management): у системі повинні бути надані механізми для відновлення сервісів і його стану, пов'язані з невдалими операціями.

– Повідомлення (Notification): безліч динамічних, розподілених сервісів повинні бути здатними виявити один одного та одержати інформацію про зміну їхнього стану.

– Керованість (Manageability): надані операції, що відносяться до керування та контролю великої кількості екземплярів класів Grid-сервісів.

– Просте хост-середовище (Simple hosting environment): просте виконавче середовище – це ряд ресурсів, розташованих у межах окремого адміністративного домена, що підтримує рідні засоби для керування сервісом: наприклад, сервер додатків J2EE, система Microsoft.NET, або кластер Linux.

Компоненти Globus, на які OGSA впливає найбільше:

– протокол GRAM (протокол розподілу та керування ресурсами Grid);

– інформаційна інфраструктура, Meta Directory Service (MDS-2), яка використовувана для виявлення інформації, реєстрації, моделювання даних і локального системного реєстру;

– GSI (The Grid Security Infrastructure, інфраструктура безпеки Grid).

Очікується, що наступні реалізації інструментарію Globus, починаючи з GT-4.0 (2005 рік) будуть засновані на архітектурі OGSA. Основні сервіси здійснять інтерфейси та визначають поведінку, описану в специфікації Grid-сервісу. Основні сервіси здійснюють як існуючі можливості Globus, такі як керування ресурсами, передачу даних та інформаційні сервіси, так і нові можливості, такі як резервування ресурсів і їх контроль. Діапазон високорівневих сервісів буде використовувати основні сервіси, щоб забезпечити керування даними, робочим навантаженням і серверами діагностики. На використання архітектури OGSA націлені також нові покоління ПГЗ gLite.

1.5.3. Агенти

Web-сервіси забезпечують функціональну сумісність, тобто ключ до Grid обчислень, а OGSA впроваджує засоби адаптації Web-сервіси до Grid. Однак, Web-сервіси не забезпечують нове рішення багатьох із проблем масштабних розподілених систем з надвеликою кількістю запитів, і при цьому вони також не забезпечують нові методики для розробки цих систем. Отже, для обслуговування широкого кола запитів використовуються інші моделі, наприклад, засновані на програмних агентах. При цьому агенти мають наступні особливості:

– Превентивність – агенти демонструють поведінку, що спрямована на рішення поставленої мети.

– Автономія – агенти діють без зовнішнього втручання та мають деякий контроль над своїми діями та внутрішнім станом.

– Соціальна здатність – агенти взаємодіють із іншими агентами, використовуючи мову комунікації агента.

– Реактивність – агенти зберігають і відповідають за своє середовище.

Обчислення, засноване на агентах, особливо добре підходить для динамічно змінного середовища, де автономія агентів дозволяє адаптувати обчислення до змінних обставин. Це є важливою властивістю для Grid-систем третього покоління. Однією з методик для досягнення вищевказаної властивості є оперативний обмін інформацією між агентами, і для реалізації цієї методики проведені відповідні дослідження. Ринкові підходи обумовлюють особливо важливі вимоги до економічності обчислювальних систем, необхідних для Grid-додатків.

На сьогодні агенти використовуються в Grid по-різному. Хоча агенти завдяки здатності до переговорів можуть значно поліпшити планування в таких масштабних системах, як Grid, сьогодні ще небагато зроблено у сфері використання агентів для задач планування. Фактично, в стадії початкових розробок нині перебувають дві системи – AgentScape та A4.

Система A4 (The Agile Architecture and Autonomous Agent system) розв’язує проблему управління ресурсами, саме використовуючи програмні агенти, які в свою чергу «спілкуються» між собою, для того, щоб знайти нові ресурси. Кожен агент знає про існування сусіднього агента, за допомогою якого і виконує свої запити для відкриття нових ресурсів. У системі A4, агенти гомогенні і складаються із 3-х основних функціональних рівнів (рис. 1.6).

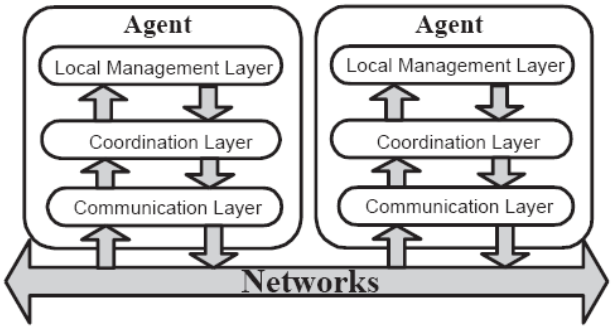


Рисунок 1.6 – Агенти системи A4

– Communication Layer (рівень комунікацій). Агенти використовують цей рівень, для того, щоб сповістити один одному про використання спільних моделей даних і протоколів зв’язку. Для цього може бути використана мова ACL (Agent Communication Language).

– Coordination Layer (рівень координації). На цьому рівні визначається, як повинен поводити себе агент з даними на рівні комунікацій.

– Local Management Layer (рівень локального управління). Цей рівень інкапсулює функції, необхідні для управління локальних сервісів. Він також забезпечує необхідною інформацією рівень координації.

Проект AgentScape забезпечує мультиагентну інфраструктуру, що може використовуватися для інтеграції та координування розподілених ресурсів у середовищі Grid. Мета системи – забезпечити мінімальне, але достатнє середовище для агентних застосувань. Модель AgentScape визначає агентів та об'єкти як головні сутності. Крім агентів, об'єктів та розташування, модель AgentScape визначає також і сервіси, які забезпечують інформацію або дії від імені агентів чи проміжного програмного забезпечення AgentScape. AgentScape надає низку компонентів, а саме ядро, сервіси каталогів та пошуку ресурсів тощо. Ця широкомасштабна розподілена агентна система спроектована для підтримки гетерогенності та інтероперабельності, полегшує розширюваність: достатньо легко будувати агентні середовища над AgentScape. Також AgentScape порівняно просто адаптується до різних операційних систем та мережних інфраструктур. По суті, AgentScape може бути досить просто інтегрована з іншими середовищами та підтримувати підхід до керування Grid-ресурсами, базований на агентах. Робота в цій галузі є незавершеною, і AgentScape лишається прототипом агентної платформи. На даний момент цей прототип реалізований на мові Java та Python, використовуючи XML-RPC для міжпроцесової комунікації. Планується, що наступний прототип реалізує захищену модель для мобільних агентних систем, P2P сервіси та іншу функціональність.

Зокрема, консорціум FIPA (Foundation for Intelligent Physical Agents) створює програмні стандарти для різнорідних і взаємодіючих агентів і систем, заснованих на агентах. В абстрактній архітектурі FIPA:

– агенти спілкуються, обмінюючись повідомленнями, які відображають динаміку змін і закодовані мовою комунікації агента;

– сервіси надають підтримку агентам, включаючи сервіси каталогів і сервіси транспортування повідомлень;

– сервіси можуть бути здійснені у вигляді агентів або у вигляді програмного забезпечення, до якого можна звернутися через мови програмування інтерфейсів (наприклад в Java, C++ або мові опису інтерфейсів).

1.5.4. Web як інформаційна інфраструктура Grid

Спочатку Інтернет у Європі просувався зусиллями CERN для розподіленого доступу до інформації в контексті е-Науки. Тоді постає питання, чи задовольняє зараз ця архітектура розподілу інформації вимогам Grid. При цьому виникають наступні питання до:

– можливості контролю версій, бо Інтернет може безупинно обновлювати сторінки без контролю версій;

– якості сервісу, оскільки вбудовані посилання можуть змінювати сервер, місце його розташування, назву або інформаційне наповнення документа, але очікуваність несуперечності посилань низька, і е-Наука може вимагати більш високої якості обслуговування;

– походженню інформації, тому що не має ніякого стандартного механізму, щоб забезпечити юридично істотний доказ, що даний документ був виданий в Інтернеті в конкретні дату та час;

– цифрового керування правами, бо е-Наука потребує специфічні функціональні можливості щодо керування цифровим інформаційним наповненням, включаючи, наприклад, керування інтелектуальною власністю та захист від копіювання;

– нагляду, тому що більша частина інфраструктури Web зосереджена на техніці доставки інформації, а не на засобах створення й керування змістом, особливо в умовах оброблення метаданих.

Web усе більше стає інфраструктурою для розподілених додатків, де скоріше відбувається обмін інформацією між програмами, ніж читання її людиною. Такий інформаційний обмін забезпечується сімейством рекомендацій XML від W3C. XML призначений для розмітки документів і не має ніякого встановленого словника тегів; вони визначені для кожного додатка й використовують Document Type Definition (DTD) або XML Schema. RDF – це стандартний спосіб вираження метаданих, особливо ресурсів на Web, хоча ним можна скористатися для опису структурованих даних взагалі. Використання XML й RDF робить можливим стандартне вираження змісту й метазмісту. З'являються додаткові набори інструментів для роботи із цими форматами, і це збільшує підтримку з боку інших інструментів. Все разом це забезпечує інфраструктуру для інформаційних систем третього покоління Grid. W3C опублікував документ [82], у якому розглянута перспективна технологія Семантичного Web, обумовлена як розширення нинішньої мережі Web, при якій інформація має чітко виражене значення, що надає кращі можливості для співробітництва людей і комп'ютерів. Головне, що несе ця технологія, – це ідея наявності даних на Web, певних і зв'язаних таким способом, що дозволяє використати їх для більше ефективного виявлення, автоматизації, інтеграції й повторного використання в різних додатках.

Таким чином, Web може досягти розкриття свого повного потенціалу, якщо стане місцем спільного використання та обробки автоматизованими інструментами і людьми, а Семантичний Web призначений зробити для надання знань те, що Web зробив для гіпертексту. DAML (DARPA Agent Markup Language) – це мовна програма розмітки агентів керування, що стартувала в 2000 році, впроваджує технології Семантичного Web для того, щоб наголосити на комунікації агентів (як обговорювалося в попередній главі). DAML розширюється XML і RDF онтологіями, потужними засобами опису об'єктів та їх відносин. Мова OIL (Ontology Interchange Language) була об'єднана з DAML, щоб сформувати DAML+OIL. W3C створив робочу групу

Web Ontology Working Group, що зосереджується на розробці мови, заснованої на DAML+OIL.

Комбінація технологій Семантичного Web з оперативними потоками інформації в значній мірі відноситься до Grid-обчислень і є новою областю. Потоки метаданих можуть бути згенеровані людьми, обладнанням або програмами – наприклад, анотація, параметри настроювання пристрою, дані, оброблені в реальному часі. Оперативні метадані в комбінації з потоками мультимедіа (такими як групове відео) піднімають вимоги до якості мережного сервісу (QoS) і тоді виникає питання, чи повинні метадані бути впроваджені. Сценарій, у якому застосовуються технології знання для розширення співробітництва, описаний в [77].

1.6. Еволюція GRID: обрії наступних поколінь

У цілому, Grid можна представити у вигляді трьохрівневої системи, що складена з рівнів обчислення/даних, інформації та знання (рис.1.7).

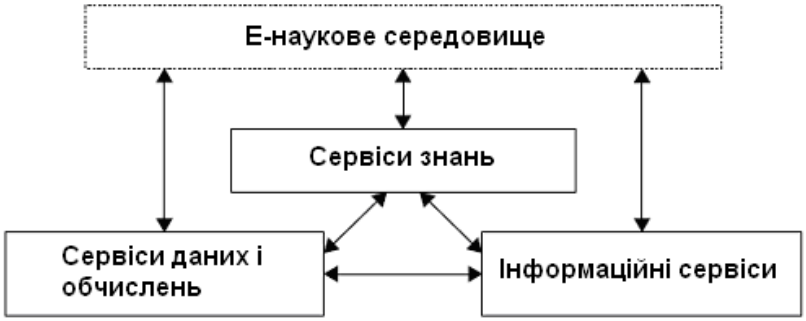


Рисунок 1.7. – Трирівнева архітектура Grid-сервісів

Семантичний Grid стане місцем, де дані будуть обладнані багатим контекстом і перетворені на інформацію. Тоді ця інформація буде розділена й оброблена віртуальними організаціями для досягнення певних цілей. Така оброблена інформація становить знання. Отже, рівень знання – ключ до наступної стадії в еволюції Grid, до повністю оснащеного Семантичного Grid.

Навіть при тому, що рівень сьогоднішніх Grid є досить високим і більшість програмного забезпечення є доступним і безпосередньо використовуваним, він усе ще має нестачу в багатьох істотних аспектах, які забезпечать ефективний доступ, поширення й використання системних ресурсів. Можна виділити наступні задачі, що вимагають подальшого розроблення:

– Інформаційні служби – механізми, які використовуються для того, щоб зберігати інформацію про ресурси, тому що Grid має потребу в наданні віддалених, швидких, надійних, безпечних і масштабованих послуг.

– Інформація про ресурси – для коректної роботи Grid будуть необхідні всі види інформації, починаючи від імені ресурсу та даних безпеки і закінчуючи прикладними вимогами та параметрами користувача. Важливо, щоб

вся ця інформація була зрозумілою, легко інтерпретованою й могла бути використаною всіма службами, які її потребують.

– Виявлення ресурсу – повинні бути механізми для визначення місцезнаходження ресурсу в межах глобально розподіленої системи, якщо є унікальне ім'я ресурсу або його характеристики. Сервіси є ресурсами. Деякі ресурси можуть зберігатися, деякі можуть бути перехідними, і деякі можуть бути створені на вимогу.

– Синхронізація й координація – повинні бути механізми для керування складною послідовністю обчислень на безлічі ресурсів. Для цього може стати необхідним опис процесу й наявність подієво-орієнтованої інфраструктури, що має на увазі планування на різних рівнях, включаючи метапланування та інформаційні потоки.

– Стійкість до відмову системи – середовища повинні справлятися з відмовою компонентів програмного забезпечення та обладнання, так само як із проблемами доступу взагалі, і повинна бути продумана обробка виняткових ситуацій, що необхідна в такий динамічній багатокористувацькій мультисистемі.

– Безпека – аутентифікація, авторизація, гарантія, і механізми ведення обліку повинні бути встановлені на місці, і вони повинні функціонувати в контексті автоматизації та масштабу, що збільшується.

– Паралельність і послідовність – потреба підтримати відповідний рівень послідовності даних у паралельному, різнорідному середовищі. Для деяких додатків достатньою може бути менш чітка послідовність.

– Продуктивність – потреба бути в змозі впоратися із глобальним доступом до ресурсів, через кешування та дублювання. Привабливим виглядає переміщення коду (або сервісу) до даних (можливо зі скриптами або мобільними агентами), але вносить ряд проблем.

– Різнорідність – потреба працювати з безліччю апаратних засобів, з різним програмним забезпеченням та інформаційними ресурсами, і зробити це через множинні організації з різними адміністративними структурами.

– Масштабованість – у системі повинна бути можливість розширити число й розмір сервісів та додатків, не вимагаючи при цьому людського втручання. Для цього потрібна автоматизація та ідеальна самоорганізація.

– Інтерфейси – потреба використовувати Grid, не дивлячись на специфічне програмне забезпечення проміжного шару(middleware) в різних інфраструктурах і платформах, що дозволить працювати в змішаному Grid середовищі.

В роботі [83] накреслені риси Grid наступного покоління (NGG), які повинні дозволяти вирішувати задачі простим чином, що не вимагає від кінцевих користувачів знання складної інфраструктури Grid, підтримувати довготривалі процеси і дані (що зберігаються протягом до 50 років і більш), забезпечувати можливість опису даних і процесів в абстрактній формі, незалежній від змін апаратних засобів і інфраструктури Grid.

Передбачається ефективна інтеграція в Grid людей – постачальників даних, експертів, осіб, що ухвалюють рішення, операторів – що реалізують

деякий процес в рамках сценарію досягнення деякої мети в Grid; ефективний пошук знань в Grid виявлення корисних знань у великих масивах даних (описах елементів самого Grid).

В галузі архітектури бажана підтримка необмеженої кількості ресурсів і різних типів пристроїв (NGG можуть складатися з мільйонів пов'язаних один з одним вузлів), при чому вузол Grid – атомарний компонент Grid, що абстрагує підтримувані ресурси і що представляє інтерфейси для доступу до їх функціональності. Вузли можуть взаємодіяти один з одним за допомогою стандартних протоколів, що масштабуються, можуть організовуватися “на льоту” в групи для надання функціональності, якою не володіє жоден з учасників групи. Отримані групи вузлів можуть використовуватися як єдине ціле клієнтами Grid. Самоорганізація вузлів дозволяє підвищити стійкості і понизити витрати на управління системою. Вузли зможуть надавати нові сервіси і використовувати поняття, незнайомі клієнтам. Семантика сервісів, функцій і понять повинна бути визначена у вигляді, доступному клієнтам. Передбачається також можливість узгодження умов постачальників ресурсів з вимогами кінцевих користувачів (Servce Level Agreements) з використанням зрозумілої обом сторонам термінології; подальший контроль за SLA і представлення елементів Grid (вузлів, користувачів) у вигляді інтелектуальних агентів, що взаємодіють один з одним за допомогою брокерів. Бажане зручне адміністрування і управління конфігурацією і підтримка декількох віртуальних організацій, що функціонують поверх загальної інфраструктури Grid.

В галузі програмування в Grid наголос робиться на більш високий рівень абстракції, бо зараз Grid-інфраструктура повністю “видно” програмістові, вимагаючи ретельного планування використання різних видів ресурсів.

Підвищена складність NGG вимагатиме наявність механізмів абстракції, що роблять прозорими операції резервування і планування ресурсів, переміщення даних, синхронізації, обробки помилок, балансування навантаження і інше.; загальних і проблемно-орієнтованих абстракцій, підтримуваних сучасними середовищами програмування. Передбачено адаптацію існуючих моделей програмування, створення нових моделей, що поєднують в собі практики розподіленого і паралельного програмування; забезпечення інтероперабельності на семантичному рівні, коли завдання користувача повинне бути переведене у вимоги до ресурсах в термінах, використовуваних в Grid, і коли використовуються метадані для опису сервісів, ресурсів і користувачів, а також обмежень, політик доступу, схемам білінгу та інше.

Потрібна глобальна інформаційна модель для загальних понять, тому що існуючі стандарти опису Grid/Web-сервісів дозволяють описати сервіс тільки в термінах операцій, що надаються ним, не передаючи семантику цих операцій і їх параметрів. Потрібна семантична об'єктна модель (онтологія) для опису операцій і даних, що дозволяє зафіксувати загальну термінологію, і засоби (автоматичною) інтеграції локальних схем з різних наочних областей в одну глобальну інформаційну схему. Потрібно виробити невеликий набір високорівневих онтологій широкого застосування – для наук, бізнесу, і т.д.

Потрібна також інформаційна модель елементів Grid, яка би інтегрувала інформацію з різнорідних джерел, що використовують різні формати даних і схеми метаданих, забезпечувала засоби перекладу між форматами даних, засоби перекладу між схемами метаданих на основі стандартних базових моделей метаданих і онтологій, гарантії збереження якості і точності даних.

Семантичні технології (Semantic Web + Grid = Sematic Grid) дозволять описувати семантичні мережі з допомогою Resource Description Framework (RDF) і онтології з допомогою Web Ontology Language (OWL), а також забезпечать автоматичне виведення логічних висновків

Очікуються істотні зміни в лояльності віртуальних організацій. Зараз це, як правило, коаліція невеликої кількості постачальників ресурсів, що пропонують зазвичай однотипні ресурси, при чому всі учасники ВО розділяють в тій чи іншій мірі цілі ВО. Відносини між ними не засновані на ринкових відносинах і попиті, а грошові розрахунки не присутні або замінені віртуальними грошима (виділений бюджет). Такі ВО часто мають вигляд “внутрішніх” ВО з міркувань безпеки, а членство в них як правило статичне, особливо постачальників ресурсів.

Передбачаються перехід до динамічно створюваних віртуальних організацій з спеціальною технологією створення і управління життєвим циклом ВО (описи ролей і правил участі, управління членством, резервування ресурсів). Формування ВО може відбуватися в рамках заснованого на ринкових принципах середовища шляхом взаємного підбору партнерів відповідно до переслідуваних кожним з учасників цілями. Легкість створення ВО повинна наблизитися до легкості створення співтовариств в Web з відповідною законодавчою базою і бізнес-практикою, електронною формою контрактів, з встановленням довіри. Прикладом може служити: створення сумісного проекту декількох організацій з базами даних, що розділяються, обчислювальними ресурсами, листами розсилки. Для досягнення якнайкращих результатів при виконанні завдань взаємодія користувача з Grid повинна бути заснована на знаннях користувача ппро Grid (настройки, переваги) і знань Grid про користувача Тобто потрібні описи і зберігання персональної інформації, а також захист цієї персональної інформації.

На рис. 1.8 показано загальний вигляд світової Grid інфраструктури, що складається з обчислювальних та ресурсів збереження даних в різних країнах, які сполучені мережами з великою швидкістю передачі даних. Товсті лінії відображають мережі з високою пропускною спроможністю, що зв'язують головні центри, тонші лінії – мережі з меншою пропускною здатністю, які з'єднують головні центри з допоміжними центрами.

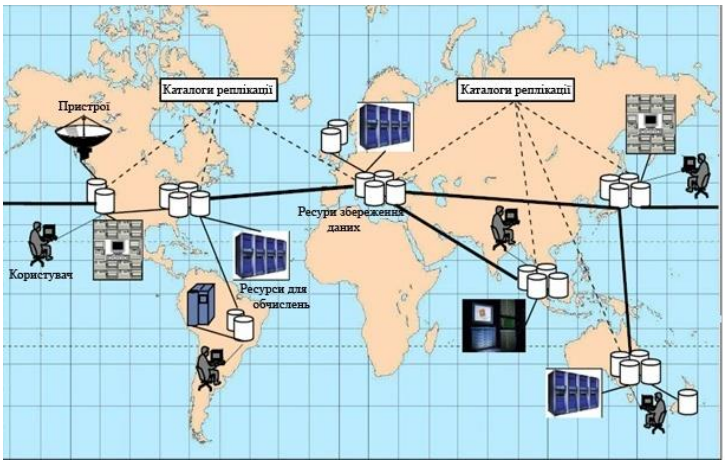


Рисунок 1.8. – Загальний вигляд світового Grid

Дані, що генеруються від приборів, експериментів або мережі датчиків зберігаються в основному вузлі збереження даних і по запиту можуть бути переміщенні до інших вузлів у всьому світі за допомогою механізму реплікації даних. Для того щоб локалізувати необхідні дані, користувачу необхідно звернутись до свого локального каталогу реплікації. Якщо користувач має необхідні права доступу, і у випадку наявності запитуваних даних на локальному репозитарії, дані вивантажуються до робочої області користувача. Якщо дані не знайдено на локальному репозитарії, тоді дані вивантажуються з віддаленого репозитарію. Для обробки дані можна передати на обчислювальний вузол, в ролі якого може бути, наприклад, кластер або суперкомп'ютер. Після обробки, результати можуть бути відправлені до засобу візуалізації, розподіленого репозитарію або до робочих станцій користувачів.

Таким чином, Grid забезпечує платформу, за допомогою якої користувачі отримують доступ до об’єднаних обчислювальних, ресурсів зберігання даних і мережевих ресурсів, для того щоб вони могли виконували свої прикладні програми для обробки розподілених даних. Це забезпечує функціональне середовище, що надає користувачам можливість проаналізували дані, спільно використовувати результати з співробітниками і використовувати інформацію про стан даних через встановлені кордони і географічні межі.

ЛЕКЦІЯ 2. АРХІТЕКТУРА GRID: РІВНІ ТА ГОЛОВНІ КОМПОНЕНТИ, ПРОТОКОЛИ ТА ІНТЕРФЕЙСИ. ПРИКЛАДИ АРХІТЕКТУРИ GRID-СИСТЕМ. ВІДКРИТА АРХІТЕКТУРА GRID-СЕРВІСІВ (OGSA) – СЕРВІСНО-ОРІЄНТОВАНИЙ ПІДХІД. ПРИКЛАДНІ ДОДАТКИ GRID: НАУКА, ПРОМИСЛОВІСТЬ, БІЗНЕС, ОСВІТА.

2.1. Архітектура Grid

Open Grid Services Architecture (OGSA) направлена на стандартизацію адресації (для сумісності), за допомогою визначення основи структури додатку GRID. По суті стандарт OGSA визначає сервіси GRID, їх можливості і те, на яких технологіях вони засновані. Проте, OGSA не розрізняє особливостей технічної сторони специфікації; метою є визначення – що є системою GRID. OGSA називають архітектурою, оскільки вона направлена на побудову і установку інтерфейсів, з яких можуть бути побудовані, системи, засновані на відкритих стандартах WSDL.

Є два основні критерії, що виділяють GRID – системи серед інших систем, що забезпечують доступ до ресурсів, що розділяються:

– GRID – система координує розрізнені ресурси. Ресурси не мають загального центру управління, а GRID – система займається координацією їх використання, наприклад, балансуванням навантаження. Тому проста система управління ресурсами кластера не є системою GRID, оскільки здійснює централізоване управління всіма вузлами даного кластера, маючи до них повний доступ. GRID – системи мають лише обмежений доступ до ресурсів, залежний від політики того адміністративного домену (організації-власника), в якому цей ресурс знаходиться.

– GRID – система будується на базі стандартних і відкритих протоколів, сервісів і інтерфейсів. Не маючи стандартних протоколів, неможливо легко і швидко підключати нові ресурси в GRID – систему, розробляти новий вигляд сервісів і так далі.

Таблиця 2.1.

Пропонований OGSA інтерфейс служб GRID

Тип порту	Операція	Опис
GridService	FindServiceData	Запит різної інформації про сервіси GRID, включаючи основну діагностичну інформацію, інформацію про інтерфейси і про особливості сервісів. Підтримка різних мов запитів
	SetTermination Time	Установка часу знищення сервісу GRID
	Destroy	Видалення служби
Notification – Source	SubscribeTo – NotificationTopic	Підписка на повідомлення про події, що відносяться до сервісів, і заснована на типі повідомлення
Notification – Sink	Deliver Notification	Виконання і асинхронна вставка повідомлення
Registry	RegisterService UnregisterService	Реєстрація додатків GRID. Анулювання реєстрації додатків GRID
Factory	CreateService	Створення нового сервісу GRID
Handle Map	FindByHandle	Повернення посилання про службу GRID, що асоціюються з їх дескрипторами

Додамо ще декілька властивостей, якими зазвичай володіють GRID – системи:

– гнучкість, тобто можливість забезпечення доступу, що розділяється, потенційно до будь-яких видів ресурсів;

– масштабованість: працездатність GRID – системи при значному збільшенні або зменшенні її складу;

– гнучка і могутня підсистема безпеки: стійкість до атак зловмисників, забезпечення конфіденційності;

– можливість контролю над ресурсами: застосування локальних і глобальних політик і квот;

– гарантії якості обслуговування;

– можливість одночасної, скоординованої роботи з декількома ресурсами.

Хоча сама технологія GRID не прив'язана до певних ресурсів, найчастіше реалізації GRID – систем забезпечують роботу з наступними типами ресурсів:

– обчислювальні ресурси – окремі комп'ютери, кластери;

– ресурси зберігання даних – диски і дискові масиви, стрічки, системи масового зберігання даних;

– мережеві ресурси;

– програмне забезпечення – яке-небудь спеціалізоване ПО.

Відзначимо різницю між технологією GRID і реалізаціями GRID – систем. Технологія GRID включає лише найбільш загальні і універсальні аспекти, однакові для будь-якої системи (архітектура, протоколи, інтерфейси, сервіси). Використовуючи цю технологію і наповнюючи її конкретним змістом, можна реалізувати ту або іншу GRID – систему, призначену для вирішення того або іншого класу прикладних завдань.

Не слід змішувати технологію GRID з технологією паралельних обчислень. В рамках конкретної GRID – системи, звичайно, можливо організувати паралельні обчислення з використанням існуючих технологій (PVM, MPI), оскільки GRID – систему можна розглядати як якийсь мета-комп´ютер, що має безліч обчислювальних вузлів. Проте технологія GRID не є технологією паралельних обчислень, в її завдання входить лише координація використання ресурсів.

Архітектура GRID визначає системні компоненти, цілі і функції цих компонент і відображає способи взаємодії компонент один з одним. Архітектура GRID є архітектурою взаємодіючих протоколів, сервісів і інтерфейсів, що визначають базові механізми, за допомогою яких користувачі встановлюють з'єднання з GRID – системою, спільно використовують обчислювальні ресурси для вирішення різного роду завдань. Архітектура протоколів GRID розділена на рівні (Рисунку 2.1), компоненти кожного з них можуть використовувати можливості компонент будь-якого з розташованих нижче рівнів. В цілому ця архітектура задає вимоги для основних компонент технології (протоколів, сервісів, прикладних інтерфейсів і засобів розробки

ПО), не надаючи строгий набір специфікацій, залишаючи можливість їх розвитку в рамках прийнятої концепції.

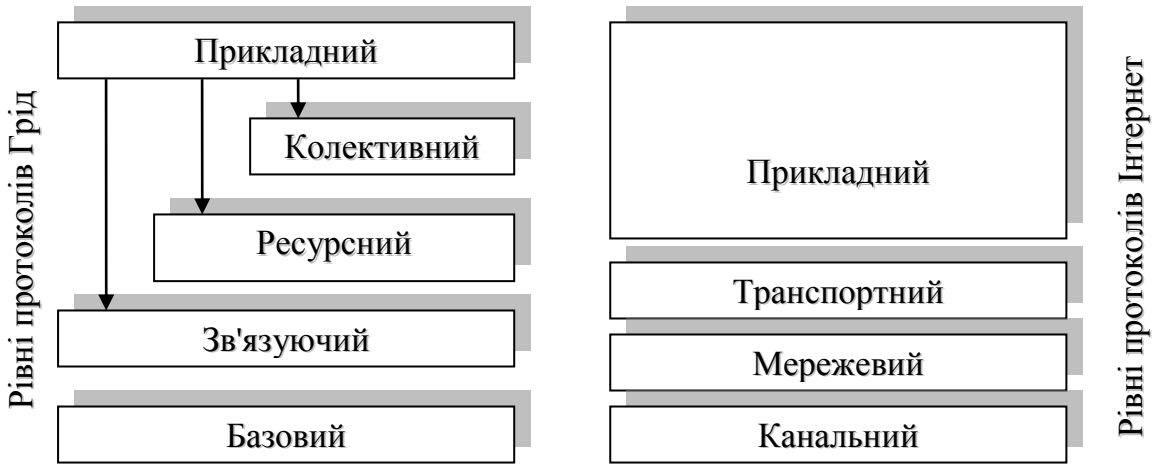


Рисунок 2.1. – Рівні архітектури протоколів Grid і їх відповідність рівням архітектури протоколів Інтернет

Інфраструктура GRID заснована на наданні ресурсів в загальне користування, з одного боку, і на використанні публічно доступних ресурсів, з іншого. У цьому плані ключове поняття інфраструктури GRID – віртуальна організація, в якій кооперуються як споживачі, так і власники ресурсів. Мотиви кооперації можуть бути різними. У існуючих GRID – системах віртуальна організація є об'єднанням (колаборацію) фахівців з деякої прикладної області, які об'єднуються для досягнення загальної мети.

GRID – система є середовищем колективного компьютинга, в якій кожен ресурс має власника, а доступ до ресурсів відкритий в режимі, що розділяється за часом і по простору, безлічі користувачів, що входять до віртуальної організації. Віртуальна організація може утворюватися динамічно і мати обмежений час існування.

Ефективний розподіл ресурсів і їх координація є основними завданнями системи GRID, і для їх вирішення використовується планувальник (брокер ресурсів). Користуючись інформацією про стан GRID – системи, планувальник визначає найбільш відповідні ресурси для кожного конкретного завдання і резервує їх для її виконання. Під час виконання завдання може запитати у планувальника додаткові ресурси або звільнити надмірні. Після завершення завдання всі відведені для неї обчислювальні ресурси звільняються, а ресурси пам'яті можуть бути використані для зберігання результатів роботи.

Важливою властивістю систем GRID є те, що користувачеві не потрібно знати про фізичне розташування ресурсів, відведених його завданню. Вся робота по управлінню, перерозподілу і оптимізації використання ресурсів лягає на планувальник і виконується непомітно для користувача. Для користувача створюється ілюзія роботи в єдиному інформаційному просторі, що володіє величезними обчислювальними потужностями і об'ємом пам'яті.

GRID є найбільш складним інформаційним середовищем, що коли-

небудь створюється людиною. Для системи такої складності дуже важлива проблема забезпечення надійного функціонування і відновлення при збоях. Людина не здатна устежити за станом тисяч різних ресурсів, що входять в GRID – систему, і з цієї причини завдання контролю над помилками покладається на систему моніторингу, яка стежить за станом окремих ресурсів. Дані про стан заносяться в інформаційні ресурси, звідки вони можуть бути прочитані планувальником і іншими сервісами, що дозволяє мати достовірну інформацію, що постійно оновлюється, про стан ресурсів.

У GRID – системах використовується складна система виявлення і класифікації помилок. Якщо помилка відбулася з вини завдання, то завдання буде зупинено, а відповідна діагностика направлена її власникові (користувачеві). Якщо причиною збою послужив ресурс, то планувальник проведе перерозподіл ресурсів для даного завдання і перезапустить його.

Збої ресурсів є не єдиною причиною відмов в GRID – системах. Через величезну кількість завдань і постійно змінної складної конфігурації системи важливо своєчасно визначати переобтяжені і вільні ресурси, проводячи перерозподіл навантаження між ними. Переобтяжений мережевий ресурс може стати причиною відмови значної кількості інших ресурсів. Планувальник, використовуючи систему моніторингу, постійно стежить за станом ресурсів і автоматично приймає необхідні заходи для запобігання перевантаженням і простою ресурсів.

У розподіленому середовищі, яким є GRID – система, життєво важливою властивістю є відсутність так званої єдиної точки збоїв. Це означає, що відмова будь-якого ресурсу не повинна приводити до збою в роботі всієї системи. Саме тому планувальник, система моніторингу і інші сервіси GRID – системи розподілені і продубльовані. Не дивлячись на всю складність, архітектура GRID розроблялася з метою забезпечити максимальну якість сервісу для користувачів. У GRID – системах використовуються сучасні технології передачі даних, забезпечення безпеки і відмовостійкої.

2.1.1 Базовий рівень

Рисунок 1.2. – Ресурси GRID

Базовий рівень (Fabric Layer) архітектури GRID описує служби, що безпосередньо працюють з ресурсами. Ресурс є одним з основних понять архітектури GRID. Ресурси можуть бути вельми різноманітними, проте, як вже

згадувалося раніше, можна виділити декілька основних типів (рисунок 1.2.):

- обчислювальні ресурси;
- ресурси зберігання даних;
- інформаційні ресурси, каталоги;
- мережеві ресурси.

Обчислювальні ресурси надають користувачеві GRID – системи (точніше кажучи, завданню користувача) процесорні потужності. Обчислювальними ресурсами можуть бути як кластери, так і окремі робочі станції. При всій різноманітності архітектури будь-яка обчислювальна система може розглядатися як потенційний обчислювальний ресурс GRID – системи. Необхідною умовою для цього є наявність спеціального програмного забезпечення, званого ПЗ проміжного рівня (middleware), що реалізує стандартний зовнішній інтерфейс з ресурсом і дозволяє зробити ресурс доступним для GRID – системи. Основною характеристикою обчислювального ресурсу є продуктивність.

Ресурси пам'яті є простором для зберігання даних. Для доступу до ресурсів пам'яті також використовується програмне забезпечення проміжного рівня, що реалізує уніфікований інтерфейс управління і передачі даних. Як і у разі обчислювальних ресурсів, фізична архітектура ресурсу пам'яті не принципова для GRID – системи, будь то жорсткий диск на робочій станції або система масового зберігання даних на сотні терабайт. Основною характеристикою ресурсу пам'яті є його об'єм.

Інформаційні ресурси і каталоги є особливим видом ресурсів пам'яті. Вони служать для зберігання і надання метаданих і інформації про інші ресурси GRID – системи. Інформаційні ресурси дозволяють структуровано зберігати величезний об'єм інформації про поточний стан GRID – системи і ефективно виконувати завдання пошуку.

Мережевий ресурс є сполучною ланкою між розподіленими ресурсами GRID – системи. Основною характеристикою мережевого ресурсу є швидкість передачі даних. Географічно розподілені системи на основі даної технології здатні об'єднувати тисячі ресурсів різного типу, незалежно від їх географічного положення.

2.1.2 Рівень зв'язку

Рівень зв'язку (Connectivity Layer) визначає комунікаційні протоколи і протоколи аутентифікації.

Комунікаційні протоколи забезпечують обмін даними між компонентами базового рівня.

Протоколи аутентифікації, ґрунтуючись на комунікаційних протоколах, надають криптографічні механізми для ідентифікації і перевірки достовірності користувачів і ресурсів.

Протоколи рівня зв'язку повинні забезпечувати надійний транспорт і маршрутизацію повідомлень, а також привласнення імен об'єктам мережі. Не дивлячись на існуючі альтернативи, зараз протоколи рівня зв'язку в GRID – системах припускають використання тільки стека протоколів TCP/IP, зокрема:

на мережевому рівні – IP і ICMP, транспортному рівні – TCP, UDP, на прикладному рівні – HTTP, FTP, DNS, RSVP. Враховуючи бурхливий розвиток мережевих технологій, в майбутньому рівень зв'язку, можливо, залежатиме і від інших протоколів.

Для забезпечення надійного транспорту повідомлень в GRID – системі повинні використовуватися рішення, що передбачають гнучкий підхід до безпеки комунікацій (можливість контролю над рівнем захисту, обмеження делегування має рацію, підтримка надійних транспортних протоколів). В даний час ці рішення ґрунтуються як на існуючих стандартах безпеки, спочатку розроблених для Інтернет (SSL, TLS), так і на нових розробках.

2.1.3. Ресурсний рівень

Ресурсний рівень (Resource Layer) побудований над протоколами комунікації і аутентифікації рівня зв'язку архітектури GRID. Ресурсний рівень реалізує протоколи, що забезпечують виконання наступних функцій:

- узгодження політик безпеки використання ресурсу;
- процедура ініціації ресурсу;
- моніторинг стану ресурсу;
- контроль над ресурсом;
- облік використання ресурсу.

Протоколи цього рівня спираються на функції базового рівня для доступу і контролю над локальними ресурсами. На ресурсному рівні протоколи взаємодіють з ресурсами, використовуючи уніфікований інтерфейс і не розрізняючи архітектурні особливості конкретного ресурсу.

Розрізняють два основні класи протоколів ресурсного рівня:

– інформаційні протоколи, які отримують інформацію про структуру і стан ресурсу, наприклад, про його конфігурацію, поточне завантаження, політику використання;

– протоколи управління, які використовуються для узгодження доступу до ресурсів, що розділяються, визначаючи вимоги і допустимі дії по відношенню до ресурсу (наприклад, підтримка резервування, можливість створення процесів, доступ до даних). Протоколи управління повинні перевіряти відповідність запрошуваних дій політиці розділення ресурсу, включаючи облік і можливу оплату. Вони можуть підтримувати функції моніторингу статусу і управління операціями.

Список вимог до функціональності протоколів ресурсного рівня близький до списку для базового рівня архітектури GRID. Додалася лише вимога єдиної семантики для різних операцій з підтримкою системи оповіщення про помилки.

2.1.4. Колективний рівень

Колективний рівень (Collective Layer) відповідає за глобальну інтеграцію різних наборів ресурсів, на відміну від ресурсного рівня, сфокусованого на роботі з окремо узятими ресурсами. У колективному рівні розрізняють загальні і специфічні (для додатків) протоколи. До загальних

протоколів відносяться, в першу чергу, протоколи виявлення і виділення ресурсів, системи моніторингу і авторизації співтовариств. Специфічні протоколи створюються для різних додатків GRID (наприклад, протокол архівації розподілених даних або протоколи управління завданнями збереження стану і тому подібне).

Компоненти колективного рівня пропонують величезну різноманітність методів сумісного використання ресурсів. Нижче приведені функції і сервіси, що реалізуються в протоколах даного рівня:

– сервіси каталогів дозволяють віртуальним організаціям виявляти вільні ресурси, виконувати запити по іменах і атрибутах ресурсів, таким як тип і завантаження;

– сервіси сумісного виділення, планування і розподілу ресурсів забезпечують виділення одного або більше ресурсів для певної мети, а також планування виконуваних на ресурсах завдань;

– сервіси моніторингу і діагностики відстежують аварії, атаки і перевантаження;

– сервіси дублювання (реплікації) даних координують використання ресурсів пам'яті в рамках віртуальних організацій, забезпечуючи підвищення швидкості доступу до даним відповідно до вибраних метрик, таких як час відповіді, надійність, вартість і т.п.;

– сервіси управління робочим завантаженням застосовуються для опису і управління багатокроковими, асинхронними, багатокомпонентними завданнями;

– служби авторизації співтовариств сприяють поліпшенню правил доступу до ресурсів, що розділяються, а також визначають можливості використання ресурсів співтовариства. Подібні служби дозволяють формувати політики доступу на основі інформації про ресурси, протоколи управління ресурсами і протоколи безпеки зв’язуючого рівня;

– служби обліку і оплати забезпечують збір інформації про використання ресурсів для контролю звернень користувачів;

– сервіси координації підтримують обмін інформацією в потенційно великому співтоваристві користувачів.

2.1.5. Прикладний рівень

Прикладний рівень (Application Layer) описує призначені для користувача застосування (додатки), що працюють в середовищі віртуальної організації. Додатки функціонують, використовуючи сервіси, визначені на рівнях, що пролягають нижче. На кожному з рівнів є певні протоколи, що забезпечують доступ до необхідних служб, а також прикладні програмні інтерфейси (Application Programming Interface – API), відповідні даним протоколам.

Для полегшення роботи з прикладними програмними інтерфейсами користувачам надаються набори інструментальних засобів для розробки програмного забезпечення (Software Development Kit – SDK). Набори інструментальних засобів високого рівня можуть забезпечувати

функціональність з одночасним використанням декількох протоколів, а також комбінувати операції протоколів з додатковими викликами прикладних програмних інтерфейсів нижнього рівня.

Звернемо увагу, що додатки на практиці можуть викликатися через достатньо складні оболонки і бібліотеки. Ці оболонки самі можуть визначати протоколи, сервіси і прикладні програмні інтерфейси, проте подібні надбудови не відносяться до фундаментальних протоколів і сервісів, необхідних для побудови GRID – систем.

2.1.6. Стандарти, що використовуються для побудови архітектури GRID

Існує декілька стандартів, використовуваних для побудови архітектури GRID. Ці стандарти утворюють базові блоки, які дозволяють посилати запити додаткам і базам даних. Ці стандарти також дозволяють розвернути програмне забезпечення, що дозволяє спростити управління бізнес – процесом. До стандартів GRID і зв'язаних з ними стандартів слід віднести:

– Комунікації «програма – програма» (SOAP, WSDL, і UDDI).

– Сумісне використання даних (мова XML).

– Передача повідомлень(SOAP, WS – Addressing, MTOM (для додатків).

– Надійна передача повідомлень (WS – Reliable Messaging).

– Управління робочим процесом (WS – Management).

– Управління транзакціями(WS – Coordination, WS – Atomic Transaction, WS – Business – Activity).

– Розподіл ресурсів (WS – RF або система ресурсних Web – сервісів).

– Забезпечення безпеки (WS – Security, WS – SecureConversation, WS – Trust, WS – Federation, Система безпечних зв'язків Kerberos для Web – сервісів.

– Обробка метаданих(WSDL, UDDI, WS – Policy).

– Orchestration (стандарти, використовувані для абстрагування бізнес – процесів від логіки додатків і джерел даних і для встановлення правив, які дозволяють бізнес – процесам взаємодіяти між собою).

– Верхній рівень управління бізнес – процесом (інженерна мова бізнес – процесу для Web сервісів – BPEL4WS).

– Події, що запускають бізнес – процес (WS – Notification).

Горизонтальна і вертикальна інфраструктура програмного забезпечення, яка необхідна для забезпечення безпеки, пошти, обміну повідомленнями, робочого потоку, колаборації, обмінів програма – програма, а також середовища сумісного використання даних може бути знайдена в інфраструктурних пропозиціях таких компаній як IBM (WebSphere), Microsoft (NET), BEA (WebLogic) і Sun (ONE). Web – сервіси і XML реалізації містяться в пропозиціях інших постачальників.

2.1.6.1. Сервіс-орієнтована архітектура

Фундаментальною концепцією OGSA є те, що сервісна архітектура складається з компонентів служб GRID, які працюють як спеціальні Web – сервіси, що надають ряд інтерфейсів, що відповідають спеціальним вимогам. SOA визначає, як взаємодіють два обчислювальні об'єкти, для того, щоб один об'єкт дав можливість другому виконати визначену роботу на користь першого. Ця робота співвідноситься з сервісом, а дії сервісу визначаються мовою описів. Кожна взаємодія самостійна і практично не залежить від іншого. Бізнес додатки створюються для автоматизації різних бизнес – процесів, але часто без здійснення в них можливості адаптуватися до потреб, що змінюються; доопрацювання бізнес процесів в даному середовищі, достатньо трудомістке завдання. Все тому, що бізнес додатки традиційно створюються як одиничні, монолітні, такі, що включають всі інструменти. Тому будь – які зміни в них достатньо дорогі і займають багато часу. У середовищі SOA, додатки створюються у вигляді набору сервісів, кожен з яких має свої завдання і властивості. У міру зміни потреб, деякі сервіси можуть додаватися, деякі видалятися або доопрацьовуватися.

Web – сервіси володіють наступними характеристиками:

– Це Інтернет додаток, що виконує спеціальні завдання і що підкоряється стандартним специфікаціям.

– Сервіс є здійснимим, він описується на XML і доступі до нього може бути здійснений за допомогою XML – повідомлень.

– Він може бути анонсований, виявлений і викликаний в розподіленому обчислювальному середовищі.

– Він не залежить від платформи або мови.

Web – сервіс є системою ПЗ, URL якого ідентифікується, а інтерфейси і зв'язки визначені і описані за допомогою XML. Він може бути виявлений іншими системами ПЗ. Ці системи, у свою чергу, можуть взаємодіяти з Web – сервісом, використовуючи XML повідомлення, що передаються за допомогою Інтернет протоколів. Мова описів Web–сервісів (WSDL) фактично є, заснованим на XML, стандартом для опису Web–сервісів. Простий протокол доступу до об'єктів (SOAP) є, заснованим на XML, стандартним мережевим протоколом для обміну повідомленнями між Web – сервісами (описи W3C).

2.1.6.2. Мова описів Web – сервісів

WSDL документ (мова описів Web – сервісів) визначає Web – сервіс, використовуючи приведені нижче основні елементи:

Таблиця 2.2.

Деякі позначення елементів у мові описів Web – сервісів

Елемент	Визначає
<portType>	Постачальники повинні показувати, чи можливо ефективно управляти процесами GRID, включаючи модель на випадок збитків
<Message>	Повідомлення, використовувані в Web – сервісе. Абстрактне визначення передаваних даних
<Types>	Типи даних, використовувані в Web – сервісе. Надає інформацію про будь – які складні типи даних, вживаних в документах WSDL. У разі використання простих типів даних цей елемент не потрібний.
<Binding>	Протоколи з'єднання, використовувані в Web – сервісе. Описує, як викликається операція, за допомогою визначення протоколу і формату даних
<Port>	Визначає одиночну крайову крапку у вигляді адреси для з'єднання, тобто крайову точку з'єднання
<Service>	Визначає адреси портів з'єднання. Служба – сукупність мережевих крайових крапок або портів

WSDL документ містить описи елементів, що складаються з блоків types, message, portType, binding, і service elements, що описане в таблиці вище. Основна структура документа WSDL виглядає таким чином:

Лістинг 2.1 – Основна структура документа WSDL

```
<definitions>
<types>
опис типів...
</types>
<message>
опис повідомлень...
</message>
<portType>
опис порту...
</portType>
<binding>
опис з'єднання...
</binding>
</definitions>
```

2.1.6.3. Web Services Inspection Language

Web Services Inspection Language (WSIL) – простий механізм виявлення Web – сервісів. WSIL – формат XML документа, створений для полегшення збору і виявлення Web – сервісів. Створений IBM і Microsoft і виданий в кінці 2001 року, WSIL є привабливим за рахунок своєї простоти, в порівнянні з UDDI, він простий і краще «піднімає» існуючі Web – сервіси. Модель WSIL децентралізована і «піднімає» існуючі Web – сервіси прямо на місці.

2.1.6.4. Universal Description, Discovery, and Integration

Universal Description, Discovery, and Integration (UDDI) – стандартний протокол опису Web – сервісів і протокол їх пошуку. Реєстр (UDDI) може містити метадані для будь – яких видів сервісів, разом з варіантами «якнайкращої практики», вже визначеними для сервісів, описаних за допомогою WSDL. За рахунок розбиття Web – сервісів на групи, що взаємодіють з категоріями і бізнес процесами, UDDI здатний ефективно шукати Web – сервіси. Специфікація UDDI визначає ієрархічну схему XML, що забезпечує модель для анонсування, перевірки і виклику інформації про Web – сервіси. Вибір ліг на XML, оскільки його формат представлення даних не залежить від платформи і відображає ієрархічні взаємозв'язки. У UDDI використовуються технології, засновані на загальних інтернет протоколах TCP/IP, HTTP, XML і SOAP. Існує 2 види UDDI реєстрів: публічні реєстри UDDI – службовці точками збору різних бізнесів, для повідомлення про їх сервіси, приватні реєстри UDDI, які роблять те ж саме але для організацій.

UDDI регістр містить наступні структурні типи даних:

– businessEntity. XML – элемент верхнього рівня в бізнес запису UDDI. businessEntity збирає дані по запиту інформації об бізнес обслуговуванні, категорії продукту або виробництва, географічному положенні, а також контактну інформацію. Він підтримує пошук по організаціях, продуктах і географічному положенні.

– businessService. Логічне продовження структури даних businessEntity і родоначальник структури bindingTemplate. businessService містить описову інформацію бізнес послуг з груп споріднених технічних послуг, включаючи ім'я групи, коротку інформацію про групу, опис технічної послуги, інформацію про категорію.

– bindingTemplate. Логічне продовження структури businessService. bindingTemplate містить дані, що відносяться до додатків, які необхідно запустити або пов'язати з Web – сервісом. Ця інформація містить URL Web – сервіса, посилання на специфікації інтерфейсу і ін.

– tModel. Містить описи специфікації Web – сервісів або систематики, які формують основу для технічних ідентифікаторів. Роль tModel полягає в наданні технічних специфікацій Web – сервісів, що дозволяє полегшити пошук Web – сервісів, сумісних з певною технічною специфікацією. Користувачі Web – сервісів можуть легко визначити інші сумісні Web – сервіси, грунтуючись на описі специфікацій в структурі tModel. Наприклад, для того, щоб послати биснес – партнеру RFP, запрошуюча служба повинна знати не тільки URL/местоположение служби, але і в якому форматі повинен бути посланий RFP, які протоколи використовувати, врахувати вимоги безпеки, яку форму відгуку має на увазі відсилання RFP.

2.1.6.5. Протокол SOAP (Simple Object Access Protocol)

SOAP – простий, заснований на XML, протокол, для обміну інформацією в децентралізованому, розподіленому середовищі. SOAP підтримує різні стилі обміну інформацією, включаючи:

– Обмін інформацією, що формується після видаленого виклику процедури. Цей тип обміну робить доступним процес запит – відповідь, в якому крайовий користувач отримує процедурне повідомлення і дає відповідь відповідним повідомленням.

– Інформаційний обмін на основі механізму обміну повідомленнями. Цей тип обміну використовують організації і додатки, яким потрібно обмінюватися бізнес – документами, послане повідомлення не має на увазі негайний відгук на нього.

SOAP характеризується:

– Протокольною незалежністю.

– Мовною незалежністю.

– Незалежністю від ОС і платформи.

– Підтримкою SOAP XML – повідомлень взаємодіючих частин (використовуючи багатоскладну структуру MIME).

Повідомлення SOAP складається з (1) SOAP конверта, який містить дві структури даних, (2) SOAP – заголовка і тіла SOAP і (3) інформації про імена, службовців для їх опису. Заголовок є необов'язковою частиною, він передає інформацію про запит, визначений в тілі SOAP. Наприклад, він може містити інформацію по безпеці, ділову інформацію або профіль користувача. Тіло містить запит Web – сервісу або відповідь на нього.

Специфікація описує структуру і тип даних при обміні повідомленнями, використовуючи XML – схему. Спосіб, в якому SOAP використовується для посилки запитів і отримання відповідей від Web – сервіса:

– Клієнт SOAP використовує документ XML, який узгоджується із специфікацією SOAP і містить запит про послугу.

– Клієнт SOAP посилає документ серверу SOAP, а той обробляє його за допомогою HTTP, HTTPS.

– Web – сервіс отримує повідомлення SOAP, направляє його, у вигляді службового запиту, додатку, що надає запрошувану послугу.

– Відгук від сервісу повертається SOAP серверу, використовуючи SOAP протокол, а це повідомлення повертається SOAP – клиенту, що послав запит.

2.2. Сервіс-орієнтована архітектура

Фундаментальною концепцією OGSA є те, що сервісна архітектура складається з компонентів служб GRID, які працюють як спеціальні Web – сервіси, що надають ряд інтерфейсів, що відповідають спеціальним вимогам. SOA визначає, як взаємодіють два обчислювальні об'єкти, для того, щоб один об'єкт дав можливість другому виконати визначену роботу на користь першого. Ця робота співвідноситься з сервісом, а дії сервісу визначаються мовою описів. Кожна взаємодія самостійна і практично не залежить від іншого. Бізнес додатки створюються для автоматизації різних бизнес – процесів, але часто без здійснення в них можливості адаптуватися до потреб, що змінюються; доопрацювання бізнес процесів в даному середовищі, достатньо трудомістке

завдання. Все тому, що бізнес додатки традиційно створюються як одиничні, монолітні, такі, що включають всі інструменти. Тому будь – які зміни в них достатньо дорогі і займають багато часу. У середовищі SOA, додатки створюються у вигляді набору сервісів, кожен з яких має свої завдання і властивості. У міру зміни потреб, деякі сервіси можуть додаватися, деякі видалятися або доопрацьовуватися.

Web – сервіси володіють наступними характеристиками:

– Це Інтернет додаток, що виконує спеціальні завдання і що підкоряється стандартним специфікаціям.

– Сервіс є здійснимим, він описується на XML і доступі до нього може бути здійснений за допомогою XML – повідомлень.

– Він може бути анонсований, виявлений і викликаний в розподіленому обчислювальному середовищі.

– Він не залежить від платформи або мови.

Web – сервіс є системою ПЗ, URL якого ідентифікується, а інтерфейси і зв'язки визначені і описані за допомогою XML. Він може бути виявлений іншими системами ПЗ. Ці системи, у свою чергу, можуть взаємодіяти з Web – сервісом, використовуючи XML повідомлення, що передаються за допомогою Інтернет протоколів. Мова описів Web–сервісів (WSDL) фактично є, заснованим на XML, стандартом для опису Web–сервісів. Простий протокол доступу до об'єктів (SOAP) є, заснованим на XML, стандартним мережевим протоколом для обміну повідомленнями між Web – сервісами (описи W3C).

2.2.1. Відкрита архітектура грід-сервісів

Все ж, для реалізації концепції «усе як ресурс», яка може бути моделлю для грід-сервісної інфраструктури, можливостей вищеназваних стандартів може виявитись недостатньо. Група дослідників, що представляла Global (Open) Grid Fourm, послідовно впроваджувала власну концепцію сервісно-орієнтовної архітектури Грід, яка, пройшовши розвиток від загального опису архітектури OGSA [9] до стандарту WSRF, згодом була реалізована в кількох масштабних грід-інфраструктурах. Сформувати рекомендації щодо реалізації грід-сервісів просто неможливо, не взявши до уваги цей цінний досвід.

Головні засади OGSA

Виходячи з того, що у Грід, як і у подібних внутрішніх IT-структурах підприємств, процес обчислень активно залучає операції зі створення та управління динамічниими групами ресурсів та сервісів. Ці групи ресурсів можуть бути різними за масштабом, тривалістю життя, постачальниками, можуть бути як однорідними, так і гетерогенними, можуть поєднуватись у різні способи. Ці особливості слугували основою для розробки відкритої архітектури грід-сервісів (OGSA).

Семантика грід-сервісу за OGSA

Як вже зазначалося, здатність до віртуалізації та поєднання різних сервісів прямо залежить від стандартних визначень інтерфейсів. Паралельно існує потребу у стандартній семантиці взаємодії сервісів, наприклад – у єдиній домовленості при оповіщенні про помилки.

З цією метою OGSA дає своє вузьке визначення грід-сервісу як веб-сервісу, що має набір добре визначених інтерфейсів та дотримується спеціальних домовленостей. Інтерфейси мають охоплювати виявлення, динамічне створення, управління тривалістю існування, оповіщення, управління сервісом, домовленості ж стосуються адресації (іменування) та розширюваності. Ці моменти складають основу концепції OGSA-сервісів.

Інтерфейси та домовленості, яким слідує сервіс, стосуються, зокрема, поведінки при управлінні тимчасовими екземплярами сервісів. Грід-середовище та ВО передбачають, що часто є необхідність у динамічному створенні нових екземплярів сервісів, які б могли управляти асоційованими з ними даними про стан. Коли ж дані про стан стають непотрібними – їх слід знищити. Прикладами таких тимчасових сутностей із власним станом є: дані сесії відеоконференції, запити до БД, операція з обробки даних, виділення мережевого каналу, сесія передачі даних, резервування обчислювальних потужностей тощо. Тимчасовість суттєво впливає на управління, іменування, пошук та використання сервісів.

Сервісна модель

Як вже зазначалося, головний лейтмотив OGSA – це «все має бути представлене як сервіс» (сервіс – сутність з мережевими можливостями, що надає функціональність через обмін повідомленнями). Обчислювальні ресурси, сховища, мережі, програми, бази даних – все це сервіси у OGSA. Прийняття єдиної загальної сервісно-орієнтованої моделі означає, що всі ці компоненти середовища віртуалізуються. Конкретніше кажучи, OGSA все представляє як грід-сервіс, називаючи ним веб-сервіс, який дотримується ряду домовленностей та підтримує стандартні інтерфейси. Базовий набір інтерфейсів, який реалізують усі OGSA-сервіси, дозволяє побудову сервісів вищого рівня на його основі.

Сервіси у OGSA характеризуються тими можливостями, які вони надають. OGSA-сервіс реалізує один або більше інтерфейсів, при чому кожен інтерфейс визначає набір операцій, що проводяться шляхом обміну визначеною послідовністю повідомлень. Ці інтерфейси відповідають portType у WSDL. Набір portTypes, які підтримуються конкретним сервісом (та деяка додаткова інформація по версіям), вказаний у елементі serviceType сервісу (розширення WSDL, введене в OGSA).

Грід-сервіси можуть підтримувати власний внутрішній стан (дані стану) протягом свого існування. Наявність стану відрізняє один екземпляр сервісу від іншого, що має той самий інтерфейс.

Прив’язка інтерфейса до протоколу може визначати семантику доставки, що стосується, приміром, надійності. Як вже було вказано, у динамічному грід-середовищі складно гарантувати, що повідомлення дійшло до адресата. З іншого боку, важливо, щоб адресат зі станом отримав не більше одного примірника повідомлення (або жодного). В такому випадку бажано було б використовувати протокол, який би реалізовував це правило. Іншим побажанням для протоколу є підтримка взаємної аутентифікації.

Сервіси OGSA можуть створюватись та знищуватись динамічно. Знищуватись явно за вказівкою, або ж неявно через системний збій. Для управління життєвим циклом сервісу OGSA визначає відповідні інтерфейси.

Оскільки OGSA-сервіси динамічні та підтримують стан, потрібен спосіб, щоб відрізняти один динамічно створений екземпляр сервісу від іншого. Тому кожний екземпляр грід-сервісу отримує унікальний ідентифікатор – Grid Service Handle (GSH), який відрізняє його від усіх інших існуючих на даний момент, в минулому чи майбутньому екземплярів. В разі, коли сервіс перезапускається зі збереженням стану, він, звичайно, може використовувати той самий GSH.

Грід-сервіси можуть оновлюватись впродовж свого існування, наприклад, буде додана підтримка нових протоколів. Тому GSH не несе жодної інформації щодо протоколів чи стану. Натомість така інформація, необхідна для взаємодії з окремим динамічним екземпляром, інкапсулюється у посилання Grid Service Reference (GSR). На відміну від GSH, який є незмінним, GSR для екзпмпляру сервісу може змінюватись впродовж його життя. GSR може мати явний термін придатності, або ж може стати неактуальним в будь-який час протягом періоду існування екземпляру, тому OGSA впроваджує механізм отримання оновленого GSR.

Результат від використання GSR, період існування якого закінчився, є невизначеним. Також слід мати на увазі, що навіть утримання дійсного GSR не гарантує доступу до екземпляру сервісу: доступ може бути обмежено локальними політиками. На додачу, екземпляр може аварійно завершитись, що, звісно, завадить його використанню через GSR.

2.3. e-Наука і Grid проекти

e-Наука (e-Science) характеризує сучасний підхід до науки, що включає підтримку розподіленої глобальної співпраці вчених за допомогою Інтернету і віртуалізації величезних сховищ даних, комп’ютерних ресурсів ї наукового обладнання.. Всі головні Grid-сервіси і інфраструктура забезпечують таку співпрацю і є вкрай потрібні суспільству.

У часи до появи Інтернету теоретичні і/або експериментальні дослідження проводили одиночки і невеликі колективи, а засобом обміну результатами були публікації статей

Зараз Інтернет і Grid дозволяють вченим проводити збір і обробку величезних масивів експериментальних даних або результатів моделювання, розроблення засобів моделювання і аналізу даних, забезпечують віддалений доступ до складних приладів і обмін інформацією в рамках розподілених, міждисциплінарних співтовариств дослідників

Спочатку Grid-технології призначалися для вирішення складних наукових, виробничих і інженерних завдань, які неможливо вирішити в розумні терміни на окремих обчислювальних установках. Проте тепер область застосування технологій Grid не обмежується тільки цими типами завдань. У міру свого розвитку Grid проникає в промисловість і бізнес, крупні

підприємства створюють Grid для вирішення власних виробничих завдань. Таким чином, Grid претендує на роль універсальної інфраструктури для обробки даних, в якій функціонує безліч служб (Grid Services), які дозволяють вирішувати не тільки конкретні прикладні завдання, але і пропонують сервісні послуги: пошук необхідних ресурсів, збір інформації про стан ресурсів, зберігання і доставка даних [1]. Область застосування Grid зараз охоплює ядерну фізику, захист навколишнього середовища, прогноз погоди і моделювання кліматичних змін, чисельне моделювання в машино- і авіабудуванні, біологічне моделювання, фармацевтику [2, 44, 45].

Існує безліч прикладів, які ілюструють вражаюче зростання кількості наукових даних, що генеруються. Додатково до прикладу проекту DAME (Distributed Aircraft Maintenance Environment [67], наведеному в розділі 1, зупинимося ще на кількох проектах з величезними об'ємами даних, наприклад, з сфери біоінформатики. Вважається що людський геном містить близько 3.2 гігабазисів, що переводиться у всього біля гігабайта інформації. Проте, якщо ми додамо сюди дані про генні послідовності в 100 000 або біля того розпізнаних протеїнах і в 32000000 амінокислотах, відповідний об'єм даних зросте до 200 гігабайт. Якщо додатково включити сюди розміри рентгенівських структур цих протеїнів то розмір даних різко зростає до декількох петабайт, і це якщо враховувати тільки по одній структурі на протеїн. Об'єм ще збільшується якщо ми врахуємо дані про можливі лікарські цілі для кожного протеїну – до 1000 наборів даних для кожного протеїну. Додатковий об'єм даних необхідний у разі дослідження генетичних варіацій людського генома. Щоб по-іншому проілюструвати проблему цих біоінформативних даних, слід звернути увагу тільки на одну з технологій, які використовуються при генерації таких даних. Розглянемо отримання рентгенівських даних за допомогою сучасного покоління електронних синхротронів. З швидкістю 1200 зображень в годину, кожна експериментальна станція генерує біля терабайта рентгенівських даних в день. Синхротрон наступного покоління – 'DIAMOND' – який зараз знаходиться у стадії розробки, генеруватиме безліч петабайт даних в рік, причому більшість з цих даних необхідно передавати і аналізувати.

З появою електронних карток пацієнтів і нововведень у сфері медичної обробки зображень, кількість медичної інформації, що зберігається в цифровому вигляді, істотно зросте. Розвиток сенсорних технологій і технологій моніторингу також внесуть істотний внесок в об'єми цифрової інформації, що зберігається.

Декілька прикладів для демонстрації суті проблеми. Компанія InSite One – американська компанія, що працює у сфері медичних зображень. Вони заявляють, що річна кількість рентгенівський зображень в США перевищує 420 мільйонів і має щорічний приріст в 12%. Типове зображення містить багато мегабайт цифрових даних і повинно зберігається як мінімум п'ять років. У Великобританії, програма e-Science зараз розглядає можливість фінансування проекту по створенню маммографічного архіву. Кожна мамограма містить 100 мегабайт даних, і повинна зберігатися з відповідними метаданими. На даний момент у Великобританії щорічно створюється близько 3 мільйонів маммограм.

У США, для порівняння, 26 мільйонів маммограм, що відповідає багатьом петабайтам даних.

Критичною проблемою для таких медичних зображень, – як і для всіх медичних цифрових даних, – є проблема точності і чистоти цих даних. Це означає, що в більшості випадків немає можливості використовувати різні методи стиснення, які могли б істотно зменшити розміри даних, що зберігалися. Іншим ключовим питанням для таких медичних даних є безпека – оскільки секретність і конфіденційність даних пацієнта є найголовнішим моментом для довіри таким технологіям.

У Великобританії, загальні вимоги до кількості даних, що зберігаються, для соціальних наук збільшилися з 400 гігабайт в 1995-му році до більш ніж терабайта в 2001. Хоча подальше зростання і прогнозується, загальний об'єм не повинен перевищити 10 терабайт до 2010 року. Архів ESRC в Ессексі, підрозділ MIMAS в Манчестері і підрозділ EDINA в Единбурзі мають досвід в управлінні архівами для соціальних наук. Підрозділи MIMAS і EDINA забезпечують доступ до статистики переписів Великобританії, постійних урядових опитів, банків макроекономічних даних, наборам цифрових карт, бібліографічним базам даних і електронним журналам. Також зараз створюються величезні історичні бази даних.

З цих прикладів очевидно, що e-Science дані, які генеруються сенсорами, супутниками, складними комп'ютерними моделюваннями, високопродуктивними пристроями, науковими фотографіями і інше скоро перевищать дані, зібрані за всю історію наукових досліджень. До недавнього моменту, комерційні бази даних були найбільшими наборами даних, що зберігаються в електронному вигляді для архівації і аналізу. Такі комерційні дані зазвичай зберігаються в Базах Даних, таких як Oracle, DB2 або SQLServer. На сьогоднішній день, розмір найбільших комерційних баз даних варіюється від десятків до сотень терабайт. Очікується, що в найближчі роки ситуація різко зміниться, і об'єм наукових даних значно перевищить об'єми комерційних систем. Цей переломних момент неминуче приведе як до нових труднощів, так і до нових можливостей. Саме з цієї причини, можливості наступного покоління Grid програмне забезпечення по доступу до даних, інтеграції даних і обєднанню даних зіграють ключову роль як для e-Science так і для e-Business[6,7,8].

Для оброблення приведених та подібних масивів даних сьогодні функціонують сотні галузевих і міждисциплінарних Grid проектів, деякі з котрих приведені в табл.2.3 [45]. Вони можуть бути цікавими для українських науковців, тому нижче проведено стислий огляд проектів, що активно користуються новими досягненнями у сфері обчислень,у контексті їх застосування до прикладних галузей

Таблиця 2.3.

Перелік відомих в світі Grid проектів

Назва проекту	**Короткий опис**	**Контактна інформація**
Проекти з фізики		
Grid Physics Network	Grid інфраструктура для обробки великих масивів даних з фізики	www.griphyn.org
Particle Physics Data Grid	Обчислювальна Grid для потреб фізики елементарних частинок	http://www.ppdg.net/
LHC Com-puting Grid	Grid інфраструктура для обчислень та збереження даних результатів експериментів на LHC (Великому Адронному Коллайдері) CERN	http://lcg.web.cern.ch/LCG
Fusion Collabo-ratory	Grid інфраструктура для фізики синтезу	http://www.fusiongrid.org/
Inter-national Virtual Datagrid-lab	Grid даних для потреб фізики та астрономії	http://www.ivdgl.org/
Проекти з астрофізики		
NEMO Project	Обчислення для підводного телескопа Черенкова, для виявлення високоенергетичних частинок	http://nemoweb.lns.infn.it/
ANTARES	Обчислення для іншого підводного телескопа, для виявлення високоенергетичних частинок	http://antares.in2p3.fr/
MAGIC	Моделювання впливів на атмосферу космічних вітрів	http://wwwmagic.mppmu.mpg.de/introduction/
Проекти з астрономії		
ESA Planck mission	Створення мікрохвильової карти неба	http://www.rssd.esa.int/index.php?project=Planck
ASTRO-GRID	Віртуальна обсерваторія (Великобританія)	http://www.astrogrid.org/
US Virtual Observa-tory	Віртуальна обсерваторія (США)	http://www.us-vo.org/

Продовження таблиці 2.3.

Назва проекту	Короткий опис	Контактна інформація
Проекти з наук про Землю		
Cooperation for Earthquake Simulation	Співпраця з моделювання та передбачення землетрусів	http://www.quakes.uq.edu.au/ACES/
Geosciences Grid	Grid для підтримки досліджень з наук про Землю	http://www.geongrid.org/
Earth-System Grid	Проект з аналізу впливів на клімат, та передбачення довгострокових кліматичних змін	http://www.earthsystemgrid.org/
Інженерія		
Geodise: Aerospace Design Optimisation	Grid система по розробці та оптимізації інженерних рішень	http://www.geodise.org/
Біохімія, медицина		
Molecular Modelling for Drug Design	Розробка віртуальної лабораторії для досліджень з молекулярної біології	http://www.gridbus.org/vlab/
Neuro Science – Brain Activity Analysis	Grid система для обробки результатів зчитування активності головного мозку	http://www.gridbus.org/neurogrid/
Bioinformatics and e-science programme	Дослідження в області стволових клітин, активності головного мозку	http://www.bbsrc.ac.uk/funding/opportunities/index.htm
CHARON	Один із підпроектів EGEE з використання Grid для обчислювальної хімії	http://egee.cesnet.cz/en/voce/Charon.html
OpenMolGrid	Відкритий проект обчислювальної Grid для молекулярної хімії та інженерії	http://www.openmolgrid.org/
Grid-Enabled Medical Simulation Services	Grid система для медичного моделювання	http://www.gemss.de/
Biomedical Informatics Research Network	Система з досліджень в галузі біоінформатики	http://www.nbirn.net/

Продовження таблиці 2.3.

Назва проекту	Короткий опис	Контактна інформація
Medical Data Manager	Система з доступу до Grid -розподілених баз медичних знань	http://rainbow.essi.fr/wiki/dokuwiki/doku.php?id=public_namespace:mdm
Міжгалузеві проекти		
DataGrid	DataGrid	http://eu-datagrid.web.cern.ch/eu-datagrid/
Datacentric Grid Project	Datacentric Grid Project	http://research.cs.queensu.ca/home/skill/datacentric.html
GRid seArch& Categorization Engine (GRACE)	GRid seArch& Categorization Engine (GRACE)	http://www.grace-ist.org/
Open grid infrastructure for science	Open grid infrastructure for science	http://www.opensciencegrid.org/
Enabling Grids for E-sciencE	Enabling Grids for E-sciencE	http://www.eu-egee.org/
Berkeley Open Infrastructure for Network Computing (BOINC)	Berkeley Open Infrastructure for Network Computing (BOINC)	http://boinc.berkeley.edu/
EUROGRID	EUROGRID	http://www.eurogrid.org/
International Grid (iGrid)	International Grid (iGrid)	http://www.isoc.org/inet99/proceedings/4a/4a_2.htm

ЛЕКЦІЯ 3. КЛАСИФІКАЦІЯ СУЧАСНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ. СИСТЕМАТИКА ФЛІНА ТА ЇЇ ДЕТАЛІЗАЦІЯ. СИСТЕМИ З ЗАГАЛЬНОЮ ПАМ'ЯТТЮ. ЗАГАЛЬНА ХАРАКТЕРИСТИКА, ПРИКЛАДИ, ПРОБЛЕМИ. СИСТЕМИ З РОЗПОДІЛЕНОЮ ПАМ'ЯТТЮ. МУЛЬТИКОМП'ЮТЕРИ. СУПЕРКОМП'ЮТЕРИ. КОМП'ЮТЕРНІ КЛАСТЕРИ, ЗАГАЛЬНА ХАРАКТЕРИСТИКА, ПРИКЛАДИ, ПРОБЛЕМИ

У процесі вирішення будь-якої задачі на паралельному комп'ютері можна виділити наступні етапи: формулювання завдання, вибір методу її рішення, фіксація алгоритму, вибір технології програмування, створення програми і, нарешті, виконання її на тому чи іншому комп'ютері. Всі ці етапи важливі і для звичайних комп'ютерів, але при використанні паралельних обчислювальних систем вони набувають особливої значущості. Будь-який паралельний комп'ютер – це ретельно збалансована система, яка може дати фантастичний результат. Такі комп'ютери спеціально проектуються для того, щоб працювати з величезною продуктивністю. Але паралельні комп'ютери не можуть працювати однаково продуктивно на будь-яких програмах. Якщо структура програми не відповідає особливостям їх архітектури, то продуктивність неминуче падає.

Зазначене невідповідність може виникнути на будь-якому етапі рішення задачі. Якщо на якомусь одному кроці ми не врахували особливостей цільового комп'ютера, то великої продуктивності на програмі завідомо не буде. Справді, орієнтація на векторно-конвеєрний комп'ютер або обчислювальний кластер з розподіленою пам'яттю в чому визначить метод розв'язання задачі. В одному випадку в програмі необхідно векторизувати внутрішні цикли, а в іншому треба думати про розпаралелюванні значних фрагментів коду. І те, й інше властивість програм визначається властивостями закладених в них методів. Не володіє обраний спосіб такими властивостями, їх не буде і в програмі, а, значить, і не буде високої продуктивності.

Паралельний комп'ютер стоїть у кінці всього ланцюжка, і тому його вплив простежується скрізь. Чим акуратніше ми проходимо кожен етап, чим більше структура програми відповідає особливостями архітектури комп'ютера, тим вище її продуктивність і тим ближче вона до його пікових показників. Всі розуміють, що досягти пікової продуктивності неможливо. Цей показник у порівнянні з реальною продуктивністю скоріше відіграє роль орієнтиру, показуючи, наскільки повно використані можливості комп'ютера при виконанні тієї чи іншої реальної програми. Проте пік продуктивності обчислюється для випадку, коли все працює з максимальним завантаженням, без конфліктів і очікувань, тобто в ідеальних умовах. У реальності ж все не так. Не так складно побудувати комп'ютер з рекордними показниками пікової продуктивності. Набагато важче запропонувати ефективний спосіб його використання, оскільки тут вже потрібно враховувати усі попередні етапи у зазначеній вище ланцюжку.

Паралельні обчислювальні системи розвиваються дуже швидко. З появою обчислювальних кластерів паралельні обчислення стали доступні багатьом. Якщо раніше паралельні комп'ютери стояли у великих центрах, то зараз кластер може зібрати і підтримувати невелика лабораторія. Вартість

кластерних рішень значно нижче вартості традиційних суперкомп'ютерів. Для їх побудови, як правило, використовуються масові процесори, стандартні мережеві технології і вільно поширюване програмне забезпечення. Якщо є бажання, мінімум коштів і знань, то принципових перешкод для побудови власної паралельної системи немає.

Інтернет. Унікальне явище нашого часу. У рамках єдиної мережі об'єднані мільйони комп'ютерів. А що, якщо їх використовувати для вирішення однієї задачі? Це буде найпотужніший паралельний комп'ютер у світі, що набагато перевершує за пікової продуктивності всі комп'ютери зі списку Top500, разом узяті. З часом Інтернет дійде до кожної квартири, надаючи всім вихід у глобальну мережу. І в глобальну обчислювальну мережу. Якщо вам потрібно буде щось порахувати, то ви підключаєтеся до мережі, даєте завдання і отримуєте результат. При цьому абсолютно не важливо, де саме ваше завдання було оброблено. Звідси і народжуються ідеї побудови метакомпьютера, що включає в себе численні обчислювальні ресурси по всьому світу.

Красивих ідей при побудові паралельних обчислювальних систем дуже багато. До складу комп'ютерів входить все більше і більше всіляких обчислювальних вузлів. Часом конструкторам доводиться розробляти унікальні рішення, щоб все це безліч пристроїв змусити злагоджено працювати разом. Це правильно, на цьому тримається прогрес обчислювальної техніки. Не потрібно тільки забувати головного. Погоня за внутрішньою узгодженістю не повинна ставати самоціллю. Будь паралельний комп'ютер є інструментом для вирішення реальних завдань. У кінцевому рахунку, з цієї точки зору і потрібно все оцінювати. Якщо внутрішня узгодженість в комп'ютері досягається в ім'я ефективного вирішення завдань, то це виправдано. Якщо ні, то відразу виникає маса питань.

3.1. Класифікація паралельних комп'ютерів і систем

Існує багато різних способів організації паралельних обчислювальних систем. Тут можна назвати векторно-конвеєрні комп'ютери, масивно-паралельні і матричні системи, комп'ютери з широким командним словом, спецпроцесори, кластери, комп'ютери з багатопотокової архітектурою і т. п. У цей же список входять і ті архітектури, які ми ще не обговорювали, наприклад, систоличні масиви або dataflow- комп'ютери. Якщо ж до подібних назв для повноти опису додати і відомості про такі важливі параметри, як організація пам'яті, топологія зв'язку між процесорами, синхронність роботи окремих пристроїв чи спосіб виконання операцій, то число різних архітектур стане і зовсім безмежним.

Чому паралельних архітектур так багато? Як вони взаємопов'язані між собою? Які основні фактори характеризують кожну архітектуру? Пошук відповіді на такі питання так чи інакше призводить до необхідності класифікації архітектур обчислювальних систем [23]. Активні спроби в цьому напрямку почалися після опублікування М. Флінном в кінці 60-х років

минулого сторіччя першого варіанту класифікації, який, до речі, використовується і в даний час.

Взагалі кажучи, при введенні класифікації можна переслідувати найрізноманітніші цілі. З точки зору головного інженера організації, де встановлюється комп'ютер, поділ комп'ютерів по потужності споживаної електроенергії, звичайно ж, буде класифікацією. Планово-фінансовий відділ більше цікавить вартість. Якщо виявиться, що установка двадцяти комп'ютерів в локальній мережі обійдеться в стільки ж, у скільки і восьмипроцесорний сервер із загальною пам'яттю, то для нього це будуть обчислювальні системи одного класу. І правильно. Що заклали в основу класифікації, такі слідства і отримуємо.

Ясно, що навести порядок у хаосі дуже важливо для кращого розуміння досліджуваної предметної області. Справді, згадаймо відкритий в 1869 році Д.І.Менделєєвим періодичний закон. Виписавши на картках назви хімічних елементів і вказавши їх найважливіші властивості, він зумів знайти таке розташування, при якому чітко простежувалася закономірність у зміні властивостей елементів, розташованих у кожному стовпці і кожному рядку. Тепер, знаючи положення елемента в таблиці, він міг з великим ступенем точності описати його властивості, не проводячи з ним ніяких безпосередніх експериментів. Іншим, справді фантастичним наслідком, стало те, що даний закон вказав на декілька "білих плям" в таблиці і дозволив передбачити не тільки існування, але і властивості поки невідомих елементів. У 1875 році французький хімік Поль Еміль Ле-кок де Буабодран, вивчаючи спектри мінералів, відкрив передбачений Менделєєвим галій і вперше детально описав його властивості. У свою чергу Менделєєв, ніколи раніше не бачив даного хімічного елемента, на підставі введеної класифікації зміг і вказати на помилку у визначенні щільності галію, і обчислити правильне значення.

Щось схоже хотілося б знайти і для архітектур паралельних обчислювальних систем. Головне питання – що закласти в основу класифікації, може вирішуватися по-різному. Однак з позицій наших досліджень класифікація повинна давати ключ до розуміння того, як вирішувати завдання ефективного відображення алгоритмів на архітектуру обчислювальних систем. У деяких випадках вводять опису класів комп'ютерів. На основі інформації про приналежність комп'ютера до конкретного класу користувач сам приймає рішення про спосіб запису алгоритму в термінах обраної технології паралельного програмування. Іноді як класифікації намагаються ввести формальний опис архітектури, використовуючи спеціальну нотацію. Такий напрям цікаве тим, що створюється сприятлива основа для побудови автоматизованих систем відображення. Справді, з одного боку, є формальний опис архітектури цільового комп'ютера, з іншого боку, є формальний опис алгоритму. Створено умови для спільного дослідження цих двох об'єктів і подальшого синтезу оптимальної програми. Правда, результат дуже сильно залежить як від якості введених описів, так і від методів їх спільного дослідження. Про ступінь рішення навіть спрощеного варіанту даної проблеми

читач може судити за якістю компіляторів, що працюють на існуючих паралельних комп'ютерах.

Не варто скидати з рахунків і той факт, що вдала змістовна класифікація може підказати можливі шляхи вдосконалення комп'ютерів. Важко розраховувати на знаходження нетривіальних " білих плям ", наприклад, у класифікації за вартістю або пікової продуктивності. Однак роздуми про можливу систематиці з точки зору простоти та технологічності програмування можуть виявитися надзвичайно корисними для визначення напрямків пошуку нових архітектур.

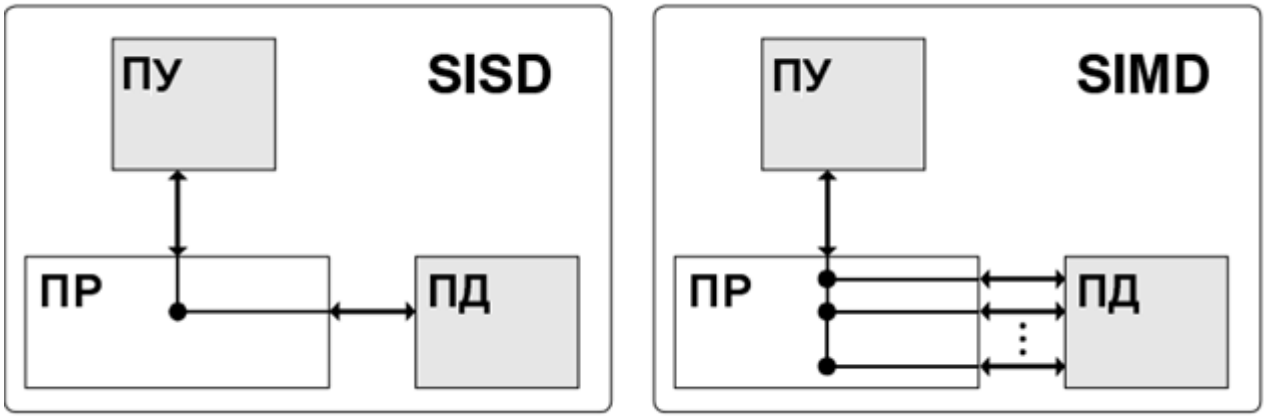


Рисунок 3.1. – Класи SISD і SIMD класифікації М. Флінна

Класифікація М. Флінна. Мабуть, найбільш ранній і найбільш відомою є класифікація архітектур обчислювальних систем, запропонована в 1966 році М. Флінном. Класифікація базується на понятті потоку, під яким розуміється послідовність команд або даних, обробляється процесором. На основі числа потоків команд і потоків даних Флінн виділяє чотири класи архітектур.

– SISD (Single Instruction stream/Single Data stream) – одиночний потік команд і одиночний потік даних (рис. 3.1). На малюнках, що ілюструють класифікацію М. Флінна, використані наступні позначення:

– ПР – це один або кілька процесорних елементів, ПУ – пристрій управління, ПД – пам'ять даних. До класу SISD відносяться, перш за все, класичні послідовні машини або, інакше, машини фоннеймановського типу, наприклад, PDP – 11 або VAX 11/ 780. У таких машинах є тільки один потік команд, всі команди обробляються послідовно один за одним і кожна команда ініціює одну скалярну операцію. Не має значення той факт, що для збільшення швидкості обробки команд і швидкості виконання арифметичних операцій може застосовуватися конвеєрна обробка: як машина CDC 6600 зі скалярними функціональними пристроями, так і CDC 7600 з конвеєрними потрапляють в цей клас.

– SIMD (Single Instruction stream/ Multiple Data stream) – одиночний потік команд і множинний потік даних (див. рис. 3.1). У архітектурах подібного роду зберігається один потік команд, що включає, на відміну від попереднього класу, векторні команди. Це дозволяє виконувати одну арифметичну операцію відразу над багатьма даними, наприклад, над елементами вектора. Спосіб

виконання векторних операцій не обмовляється, тому обробка елементів вектора може проводитися або процесорної матрицею, як у ILLIAC IV, або за допомогою конвеєра, як, наприклад, в машині Сгау -1.

– MISD (Multiple instruction stream/Single Data stream) – множинний потік команд і одиночний потік даних (рис. 3.2). Визначення увазі наявність в архітектурі багатьох процесорів, що обробляють один і той же потік даних. Проте ні Флінн, ні інші фахівці в галузі архітектури комп'ютерів до цих пір не змогли представити переконливий приклад реально існуючої обчислювальної системи, побудованої на даному принципі. Ряд дослідників відносять конвеєрні машини до даного класу, однак це не знайшло остаточного визнання в науковому співтоваристві. Будемо вважати, що поки даний клас порожній.

– MIMD (Multiple instruction stream/ Multiple Data stream) – множинний потік команд і множинний потік даних (див. рис. 3.2). Цей клас припускає, що в обчислювальній системі є кілька пристроїв обробки команд, об'єднаних в єдиний комплекс і працюючих кожне зі своїм потоком команд і даних.

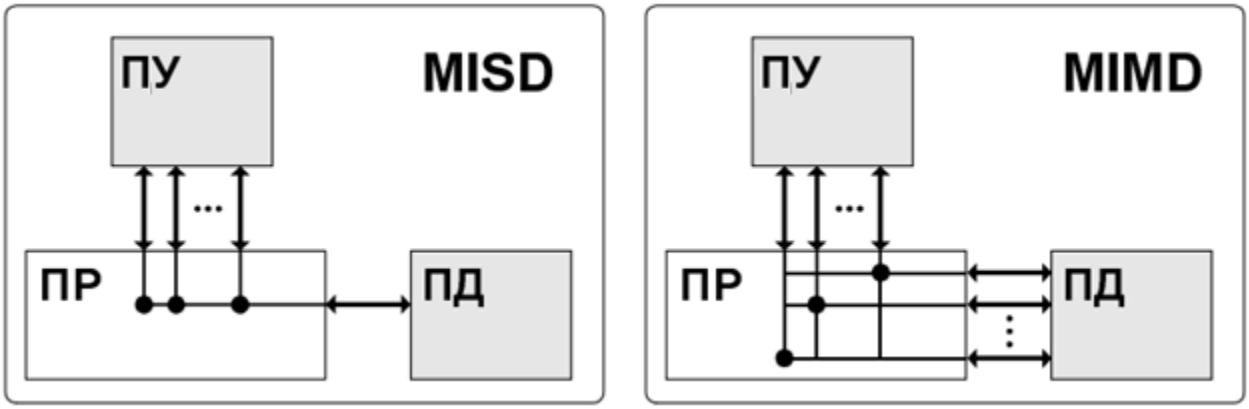


Рисунок 3.2. – Класи MISD і MIMD клас класифікації М. Флінна

Отже, що ж собою представляє кожен клас? У SISD, як уже говорилося, входять однопроцесорні послідовні комп'ютери типу VAX 11/ 780. Однак багатьма критиками помічено, що в цей клас можна включити і векторно-конвеєрні машини, якщо розглядати вектор як одне неподільне дане для відповідної команди. У такому випадку в цей клас потраплять і такі системи, як Сгау -1, СУВЕК. 205, машини сімейства ріпаком УР і багато інших.

Безперечними представниками класу SISD вважаються матриці процесорів: ILLIAC, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 і т.п. У таких системах єдине управляючий пристрій контролює безліч процесорних елементів. Кожен процесорний елемент одержує від пристрою керування в кожен фіксований момент часу однакову команду і виконує її над своїми локальними даними. Для класичних процесорних матриць ніяких питань не виникає. Однак у цей же клас можна включити і векторно – конвеєрні машини, наприклад, Сгау -1. У цьому випадку кожен елемент вектора треба розглядати як окремий елемент потоку даних.

Клас M1ME надзвичайно широкий, оскільки включає в себе все можливі мультипроцесорні системи: Cm’, C.mmp, Cray Y-MP, Denelcor HEP, BBN Butterfly, Intel Paragon, Cray T3D і багато інших. Цікаво те, що якщо конвеєрну обробку розглядати як виконання послідовності різних команд (операцій щаблів конвеєра) немає над одиночним векторним потоком даних, а над множинним скалярним потоком, то всі розглянуті вище векторно-конвеєрні комп'ютери можна розташувати і в даному класі.

Запропонована схема класифікації аж до теперішнього часу є самою застосовуваної при початковій характеристиці того чи іншого комп'ютера. Якщо говориться, що комп'ютер належить класу SIMD або MIMD,то відразу стає зрозумілим базовий принцип його роботи, і в деяких випадках цього буває достатньо. Однак видно і явні недоліки. У частності, деякі заслуговують на увагу архітектури, наприклад dataflow і векторно-конвеєрні машини, чітко не вписуються в дану класифікацію. Інший недолік – це надмірна заповненість класу MIMD. Необхідно засіб, більш вибірково систематизуюче архітектури, які за Флінном потрапляють в один клас, але зовсім різні за кількістю процесорів, природі і топології зв'язку між ними, за способом організації пам'яті і, звичайно ж, за технологією програмування.

Класифікація Р.Хокні. Р. Хокні розробив свій підхід до класифікації для більш детальної систематизації комп'ютерів, які потрапляють в клас MIMD з систематики М. Флінна. Як зазначалося вище, клас MIMD надзвичайно широкий і об'єднує ціле безліч різних типів архітектур. Намагаючись систематизувати архітектури усередині цього класу, Р. Хокні отримав ієрархічну структуру, представлену на рис. 3.3.

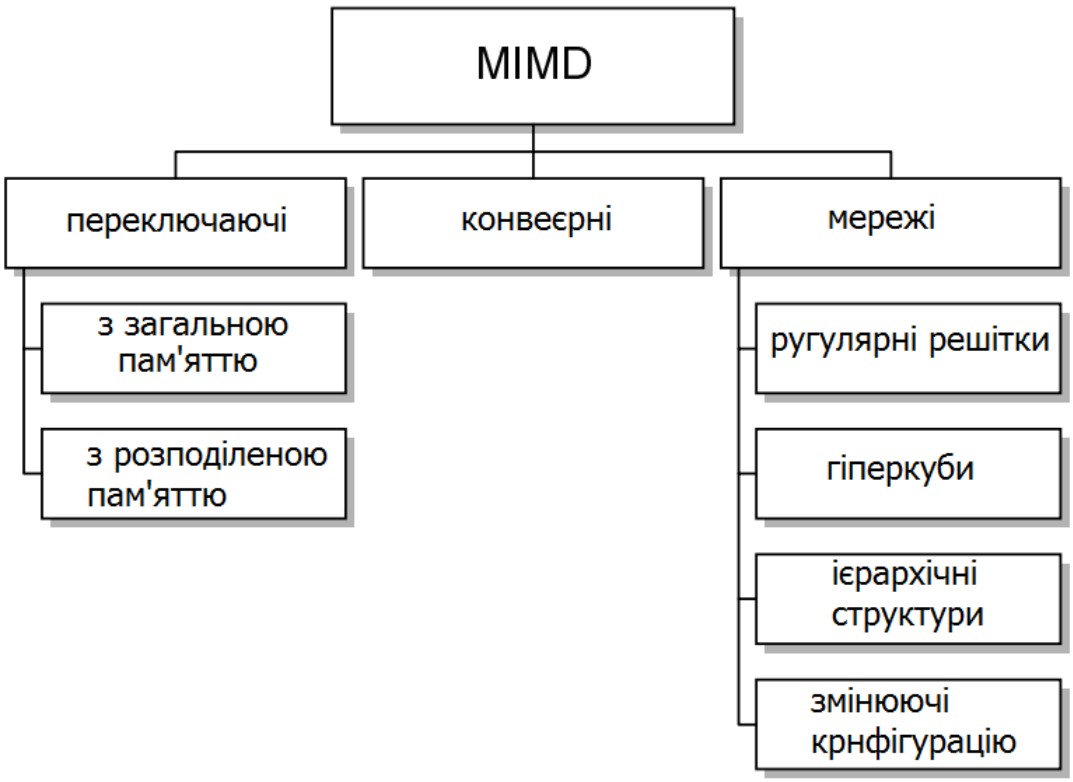


Рисунок 3.3. – Додаткова класифікація Р. Хокні класу MIMD

Основна ідея класифікації полягає в наступному. Множинний потік команд може бути оброблений двома способами: або одним конвеєрним пристроєм обробки, що працює в режимі поділу часу для окремих потоків, або кожен потік обробляється своїм власним пристроєм. Перша можливість використовується в MIMD-компютерах, які автор називає конвеєрними. Сюди можна віднести, наприклад, процесорні модулі в Denelcor HEL або комп'ютери сімейства Тега MTA.

Архітектури, що використовують другу можливість, у свою чергу, знову діляться на два класи. У перший клас потрапляють MIMD-комп'ютери, в яких можлива прямий зв'язок кожного процесора з кожним, реалізована за допомогою перемикача. У другому класі знаходяться MIMD-комп'ютери, в яких прямий зв'язок кожного процесора можлива тільки з найближчими сусідами по мережі, а взаємодія віддалених процесорів підтримується спеціальною системою маршрутизації.

Серед MIMD-машин з перемикачем Хокні виділяє ті, в яких вся пам'ять розподілена серед процесорів як їх локальна пам'ять (наприклад, PASM, PRINGlE, IBM SP2 без SMP-вузлів). У цьому випадку спілкування самих процесорів реалізується за допомогою складного перемикача, що становить значну частину комп'ютера. Такі машини звуться MIMD-машин з розподіленою пам'яттю. Якщо пам'ять це розділяється ресурс, доступний всім процесорам через перемикач, то MIMD-машини є системами із загальною пам'яттю (BBN Butterfly, Cray C90). Відповідно до типу перемикачів можна проводити класифікацію і далі: простий перемикач, багатокаскадний перемикач, загальна шина і т. п. Багато сучасні обчислювальні системи мають як загальну пам'ять, що розділяється, так і розподілену локальну. Такі системи автор розглядає як гібридні MIMD з перемикачем.

При розгляді MIMD -машин з мережевою структурою вважається, що всі вони мають розподілену пам'ять, а подальша класифікація проводиться відповідно до топології мережі: зіркоподібна мережа (ICAP), регулярні решітки різної розмірності (Intel Paragon, Cray T3D), гіперкуби (NCube, Intel і PSC), мережі з ієрархічною структурою, такий як дерева, піраміди, кластери (Cm’, CEDAR) і, нарешті, мережі, що змінюють свою конфігурацію.

Зауважимо, що якщо архітектура комп'ютера спроектована з використанням декількох мереж з різною топологією, то, по всій видимості, за аналогією з гібридними MIMD – машинами з перемикачами, їх варто назвати гібридними мережними MIMD – машинами, а використовують ідеї різних класів – просто гібридними MIMD – машинами. Типовим представником останньої групи, зокрема, є комп'ютер Connection Machine 2, що має на зовнішньому рівні топологію гіперкуба, кожен вузол якого є кластером процесорів з повним зв'язком.

*Класифікація Т. Фенга***.** У 1972 році Т.Фенг запропонував класифікувати обчислювальні системи на основі двох простих характеристик. Перша – число *n* біт у машинному слові, оброблюваних паралельно при виконанні машинних інструкцій. Практично у всіх сучасних комп'ютерах це число збігається з довжиною машинного слова. Друга характеристика дорівнює числу слів *m*,

оброблюваних одночасно даної обчислювальної системою. Трохи змінивши термінологію, функціонування будь-якого комп'ютера можна представити як паралельну обробку я бітових шарів, на кожному з яких незалежно перетворюються *m* біт. Спираючись на таку інтерпретацію, другу характеристику називають шириною бітового шару.

Кожну обчислювальну систему С можна описати парою чисел (*n, m*). Твір *P=n'm* визначає інтегральну характеристику потенціалу паралельності архітектури, яку Фенг назвав максимальним ступенем паралелізму обчислювальної системи. По суті, дане значення є не що інше, як пікова продуктивність, виражена в інших одиницях. У період появи даної класифікації, а це початок 70- х років минулого сторіччя, ще здавалося можливим перенести поняття пікової продуктивності як універсального засобу порівняння і описи потенційних можливостей комп'ютерів з традиційних послідовних машин на паралельні. Розуміння того факту, що пікова продуктивність сама по собі не настільки важлива, прийшло пізніше, і даний підхід відображає, природно, ступінь осмислення специфіки паралельних обчислень того часу.

Розглянемо комп'ютер Advance Scientific Computer фірми Texas Instruments (TI ASC). В основному режимі він працює з 64- розрядним словом, причому всі розряди обробляються паралельно. Арифметико – логічний пристрій має чотири одночасно працюючих конвеєра, що містять по вісім сходинок. При такій організації 4x8 =32 слова можуть оброблятися одночасно (тобто 32 біта в кожному бітовому шарі), і значить комп'ютер TI ASC може бути представлений у вигляді (64, 32).

На основі введених понять всі обчислювальні системи можна розділити на чотири класи.

– Розрядно-послідовні, послівно – послідовні *(n= m* = 1). У кожен момент часу такі комп'ютери обробляють тільки один двійковий розряд. Представником даного класу служить давня система MINIMA з природним описом (1, 1).

– Розрядно-паралельні, послівно – послідовні *(n*> 1, *m* = 1). Більшість класичних послідовних комп'ютерів, так само як і багато обчислювальні системи, що використовуються зараз, належать до даного класу: IBM 701 з описом (36, 1), PDP -11 з описом (16, 1), IBM 360/ 50 і VAX 11 / 780 – обидві з описом (32, 1).

– Розрядно-послідовні, послівно – паралельні *(n*= 1, *m* > 1). Як правило обчислювальні системи даного класу складаються з великого числа однорозрядних процесорних елементів, кожен з яких може незалежно від інших обробляти свої дані. Типовими прикладами служать STARAN (1, 256) і MPP (1, 16384) фірми Goodyear Aerospace, прототип відомої системи ILLIAC IV комп'ютер SOLOMON (1, 1024) і ICL DAP (1, 4096).

– Розрядно-паралельні, послівно – паралельні *(n*> 1, *m* > 1). Переважна більшість паралельних обчислювальних систем, обробляючи одночасно *m* х п двійкових розрядів, належить саме до цього класу: ILLIAC IC

(64, 64), T1 ASC (64, 32), C.mmp (16, 16), CDC 6600 (60, 10), BBN Butterfly GP1000 (32, 256).

Недоліки запропонованої класифікації досить очевидні і пов'язані зі способом обчислення ширини бітового шару т. По суті Фенг не чинить ніякої відмінності між процесорними матрицями, векторно – конвеєрними і багатопроцесорними системами. Не робиться акцент на тому, за рахунок чого комп'ютер може одночасно обробляти більше одного слова: множинності функціональних пристроїв, їх конвеєрних або ж якогось числа незалежних процесорів. Якщо в системі N незалежних процесорів мають кожен по P конвеєрних функціональних пристроїв з довжиною конвеєра *L,* то для обчислення ширини бітового шару треба просто знайти твір N'F'L

Звичайно ж, спираючись на дану класифікацію, досить важко, а іноді й просто неможливо, усвідомити специфіку тієї чи іншої обчислювальної системи. Перевагою ж можна вважати введення єдиної числової метрики для всіх типів комп'ютерів, що дозволяє порівняти будь-які два комп'ютера між собою.

Класифікація В. Хендлера. В основу класифікації В. Хендлер закладає явне опис можливостей паралельної і конвеєрної обробки інформації обчислювальної системою. При цьому він навмисно не розглядає різні способи зв'язку між процесорами і блоками пам'яті, а вважає, що комунікаційна мережа може бути потрібним чином сконфигурирована і буде здатна витримати передбачувану навантаження.

Запропонована класифікація базується на розходженні між трьома рівнями обробки даних у процесі виконання програм:

– рівень виконання програми; спираючись на лічильник команд і деякі інші регістри, пристрій управління (ПУ) виробляє вибірку і дешифрацію команд програми;

– рівень виконання команд; арифметико – логічний пристрій комп'ютера (АЛП) виконує команду, видану йому пристроєм управління;

– рівень бітової обробки; всі елементарні логічні схеми процесора (ЕЛС) розбиваються на групи, необхідні для виконання операцій над одним двійковим розрядом.

Подібна схема виділення рівнів припускає, що обчислювальна система містить якесь число процесорів, кожен зі своїм пристроєм управління. Кожен пристрій управління пов'язане з декількома арифметико – логічними пристроями, виконуючими одну і ту ж операцію в кожен конкретний момент часу. Нарешті, кожне АЛУ об'єднує кілька груп елементарних логічних схем, асоційованих з обробкою одного двійкового розряду (число груп ЕЛС є не що інше, як довжина машинного слова). Якщо на якийсь час не розглядати можливість конвейєризації, то число пристроїв управління *k*, число арифметико – логічних пристроїв *d* в кожному пристрої управління і число груп ЕЛС *w* в кожному АЛУ складуть трійку для опису даної обчислювальної системи *C:t(C)=(k, d, w).*

У таких позначеннях опису деяких добре відомих обчислювальних систем будуть виглядати наступним чином:

t(MINIMA) = (1,1,1,);
t(IBM 701) = (1,1,36);
t(SOLOMON) = (1,1024,1);
t(ILLIAC IV) = (1,64,64);
t(STARAN) = (1,8192,1) – у повній конфігурації;
t(C.mmp) = (16,1,16) – основний режим роботи;
t(PRIME)= (5,1,16);
t(BBN Butterfly GP1000)= (256,1,32).

Незважаючи на те, що перерахованим системам притаманний паралелізм різного роду, він без особливих зусиль може бути віднесений до одного з трьох виділених рівнів.

Тепер можна розширити можливості опису, допустивши можливість конвеєрної обробки на кожному з рівнів (рис. 3.4). Справді, конвеєрний на самому нижньому рівні, тобто на рівні ЕЛС, це конвеєрних функціональних пристроїв. Якщо функціональний пристрій обробляє w-розрядні слова на кожній з w' ступенів конвеєра, то для характеристики паралелізму даного рівня природно розглянути твір Знак "x" будемо використовувати на кожному рівні, щоб відокремити число, що представляє ступінь паралелізму, від числа ступенів в конвеєрі. Комп'ютер TI ASC має чотири конвеєрних пристрої по вісім ступенів в кожному для обробки 64- розрядних слів, отже, він може бути описаний так:

$$t(\text{TI ASC}) = (1, 4, 64 \times 8).$$

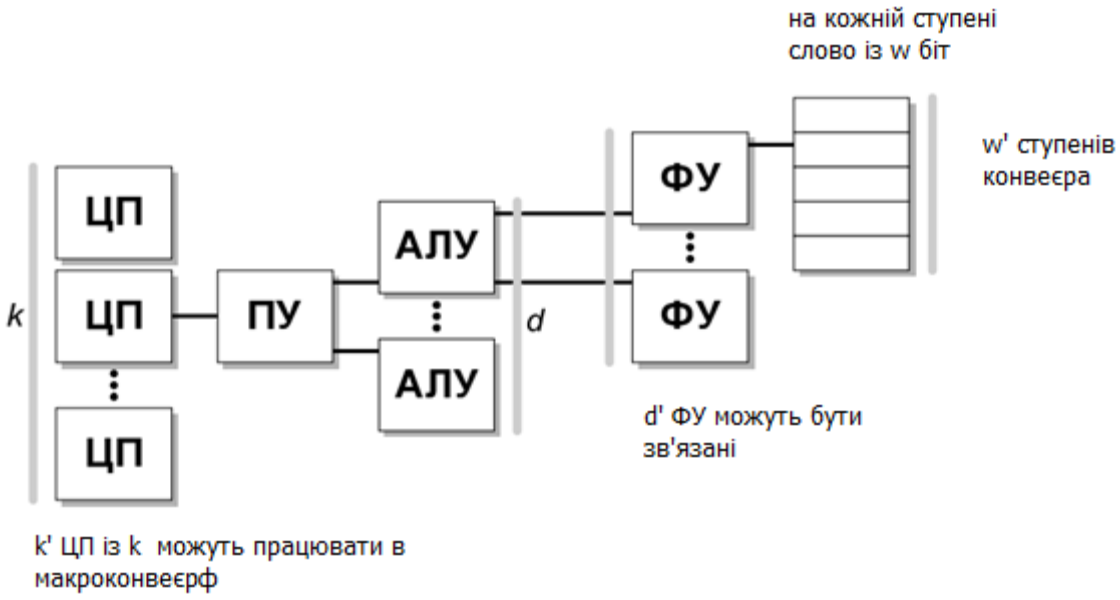


Рисунок 3.4. – Рівні паралелізму в класифікації В. Хендлера

Наступний рівень конвеєрної обробки – це конвеєризація на рівні команд. Передбачається, що в обчислювальній системі є кілька функціональних пристроїв, які можуть працювати одночасно в режимі конвеєра (для позначення даної можливості часто використовують інший термін – зачеплення

функціональних пристроїв). Класичним прикладом цього служать векторно – конвеєрні комп'ютери фірми Cray Research. Іншим прикладом є машина CDC 6600, що містить десять незалежних послідовних функціональних пристроїв, здатних подавати результат своєї роботи на вхід іншим функціональним пристроям *t*(CDC 6600) = (1,1 x 10,60) (описаний тільки центральний процесор без урахування керуючих і периферійних підсистем).

Нарешті, нам залишилося розглянути конвеєризацію на самому верхньому рівні, відому як макроконвеєра. Потік даних, проходячи через один процесор, надходить на вхід іншому. Комп'ютер PEPE, маючи фактично три незалежних системи з 288 пристроїв, описується так:

$$t(\text{Pepe}) = (1 \text{ x } 3, 288, 32).$$

Після розширення трирівневої моделі паралелізму засобами опису потенційних можливостей конвейєризації кожна трійка

$$t(\text{C}) = (k \text{ x } k', d \text{ x } d', w \text{ x } w')$$

інтерпретується так:

– k – число процесорів (кожен зі своїм ПУ), що працюють паралельно;

– k' – глибина макроконвеєра з окремих процесорів;

– d – Число АЛП в кожному процесорі, що працюють паралельно;

– d' – Глибина конвеєра з функціональних пристроїв АЛП;

– w – число розрядів у слові, оброблюваних в АЛУ паралельно;

– w' – число ступенів в конвеєрі функціональних пристроїв АЛП.

Очевидний зв'язок між класифікацією Фенга і класифікацією Хендлера: для отримання максимального ступеня паралелізму в термінах Фенга треба знайти добуток всіх шести величин в описі Хендлера. Тут же зауважимо, що, заклавши в основу своєї схеми явне вказівку на присутній паралелізм і можливу конвеєризацію, Хендлер одразу знімає деякі питання, характерні для схем Флінна і Фенга, принаймні, в плані опису векторно – конвеєрних машин.

На додаток до викладеного способу опису архітектур Хендлер пропонує використовувати три операції, які, будучи застосованими до трійкам, дозволять описати, по-перше, складні структури з підсистемами введення / виведення, хост- комп'ютером або якимись іншими особливостями, і, по-друге, можливі режими функціонування обчислювальних систем, підтримувані для оптимальної відповідності структурі програм.

Перша операція (x) в якомусь сенсі відображає конвеєрний принцип обробки і передбачає послідовне проходження даних спочатку через перший її аргумент – підсистему, а потім через другий. Зокрема, опис комп'ютера CDC 6600 можна уточнити таким чином:

$$t(\text{CDC 6600}) = (10,1,12) \text{ x } (1,1 \text{ x } 10,60),$$

де перший аргумент відображає існування десяти 12- розрядних периферійних процесорів і той факт, що будь-яка програма повинна спочатку бути оброблена одним з них і лише після цього передана центральному процесору для виконання. Аналогічно можна отримати опис машини PEPE, беручи до уваги, що в якості хост- комп'ютера вона використовує CDC 7600:

$$t(\text{Pepe}) = t(\text{CDC 7600})x(1 \text{ x } 3, 288, 32) = (15,1,12)x(1,1x9,60)x(1x3, 288,32).$$

Потік даних послідовно проходить через три підсистеми, що ми і відобразили, з'єднавши їх знаком "x".

Зауважимо, що всі підсистеми останнього прикладу досить складні і по даному опису можуть представлятися по- різному. Щоб внести більшу ясність, аналогічно операції конвеєрного виконання, Хендлер вводить операцію паралельного виконання (+), фіксуючу можливість незалежно використання процесорів різними завданнями, наприклад:

$$t(4,d,w) = [(1,d,w) + (1,d,w) + (1, d,w) + (1,d,w)].$$

У разі CDC 7600 уточнена запис виду:

$$(15, 1, 12)x(1, 1 \text{ x } 9, 60) = [(1, 1, 12) +... + (1, 1, 12)] \{ 15 \text{ раз } \}x(1, 1 \text{ x } 9, 60)$$

говорить про те, що кожна задача може захопити свій периферійний процесор, а потім одна за однією вони будуть надходити в центральний процесор.

І, нарешті, третя операція – операція альтернативи (v), показує можливі альтернативні режими функціонування обчислювальної системи. Наприклад, комп'ютер C.mmp може бути запрограмований для використання в трьох принципово різних режимах:

$$t\ (\text{C.mmp}) = (16,1,16)\ \text{V}\ (1 \text{ x } 16, 1,16)\ \text{V}\ (1, 16, 16).$$

Класифікація Л. Шнайдера. У 1988 році Л. Шнайдер запропонував виділити етапи вибірки і безпосередньо виконання в потоках команд і даних. Саме поділ потоків на адреси і їх вміст дозволило описати такі раніше "незручні" для класифікації архітектури, як комп'ютери з довгим командним словом, систолические масиви і цілий ряд інших.

Введемо необхідні для подальшого викладу поняття і позначення. Назвемо потоком посилань S деякої обчислювальної системи кінцеве безліч нескінченних послідовностей пар:

$$S = \{ (a_1 <t_1 >) (a_2 < t_2>), (b_1< u_1 >) (b_2< u_2 >), (c_1<v_1 >) (c_2 < v_2 >) \},$$

де перший компонент кожної пари – це невід'ємне ціле число, зване адресою, другий компонент – це набір з n невід'ємних цілих чисел, які

називаються значеннями, причому *n* однаково для всіх наборів всіх послідовностей. Наприклад, пара $(b_2 < u_2 >)$, визначає адресу b_2 і значення u_2. Якщо значення розглядати як команди, то з потоку посилань отримаємо *потік команд I*; якщо ж значення інтерпретувати як дані, то відповідний потік – це *потік даних D.*

Інтерпретація введених понять дуже проста. Елементи кожної послідовності це адреса і його вміст, обиране з пам'яті (або записують в пам'ять). Послідовність пар адресу – значення можна розглядати як історію виконання команд або переміщення даних між процесором і пам'яттю комп'ютера під час виконання програми. Число інструкцій, яке даний комп'ютер може виконувати одночасно, визначає число послідовностей в потоці команд. Аналогічно, число різних даних, яке комп'ютер може обробити одночасно, визначає число послідовностей в потоці даних.

Нехай S довільний потік посилань. *Послідовність адрес* потоку S, що позначається Sa, це послідовність наборів, сформована з адрес кожної послідовності з S:

$$Sa = < a1\ b1\ c1 >, < a2\ b2\ c2 >.$$

Послідовність значень потоку S, що позначається Sw, це послідовність наборів, сформована зі значень кожної послідовності з S:

$$Sw = <t1\ u1\ v1 >, <t2\ u2\ v2 >.$$

Якщо Sx – послідовність елементів, де кожен елемент є набором з *n* чисел, то для позначення " ширини " послідовності будемо користуватися позначенням:

$$w(Sx)=n$$

З визначень Sa, Sv та w випливає, що якщо S це потік посилань зі значеннями з *n* чисел, w(Sa) = |S| і, де w(Sv) = n|S|, де |S| позначає потужність множини S.

Кожну пару (*I,D*) з потоком команд *I* і потоком даних *D* називатимемо обчислювальним шаблоном, а всі комп'ютери будемо розбивати на класи в залежності від того, який шаблон вони можуть виконати. Справді, комп'ютер може виконати шаблон (*I,D*), якщо він в змозі:

- видати *w(Ia)* адрес команд для одночасної вибірки з пам'яті;
- декодувати і проінтерпретувати одночасно *w(Iv)*команд;
- видати одночасно *w(Da*) адрес операндів;
- виконати одночасно *w(Dv*)операцій над різними даними.

Якщо всі ці умови виконані, то комп'ютер може бути описаний як $I_{w(Ia)w(Iv)}D_{w(Da)w(Dv)}$ Розглянемо класичну послідовну машину. Відповідно до класифікації Флінна, вона потрапляє в клас SISD, отже $|I| = |D| = 1$ і $w(Ia)=w(Da)= 1$. Через те, що в подібного роду комп'ютерах команди

декодуються послідовно, слідує рівність *w(Iv)* = 1, а послідовне виконання команд дає *w(Dv)* = 1. Тому опис однопроцесорної машини з Фоннеймановская архітектурою буде виглядати так: $I_{1,1}D_{1,1}$

Тепер візьмемо дві машини з класу SIMD: Goodyear Aerospace MPP та ILLIAC IV, причому не будемо брати до уваги різницю в способах обробки даних окремими процесорними елементами. Єдиний потік команд означає | *I* | = 1 для обох машин. Аналогічно послідовної машині, для потоку команд отримуємо рівність і *w(Ia)=w(Iv)*= 1. Далі, згадаємо, що для доступу до операндів пристрій управління MPP розсилає один і той же адресу всім процесорним елементам, тому в цій термінології MPP має єдину послідовність у потоці даних, тобто |*D*|=1. Проте потім вибірка даних з пам'яті і наступна обробка здійснюються в кожному процесорному елементі, тому *w(Dv)* = 16 384, а вся система MPP може бути описана $I_{1,1}D_{1,16384}$ В ILLIAC IVпристрій управління, так само, як і в MPP, розсилає один і той же адресу всім процесорним елементам, проте кожен з них може отримати свою унікальну адресу, додаючи вміст локального індексного регістра. Це означає, що |*D*| = 64 і в системі присутні 64 потоку адрес даних, що визначають одиночні потоки операндів, тобто *w(Da)=w(Dv)*= 64. Підсумовуючи сказане, приходимо до опису ILLIAC IV: $I_{1,1}D_{64,64}$

Для більш детальної класифікації Шнайдер вводить три визначених значення, які можуть приймати величини *w(Ia), w(Iv), w(Da) і w(Dv)*:

- *S* – значення дорівнює 1;
- *c* – значення від 1 до деякої (невеликий) константи;
- *m* – значення від 1 до довільно великого кінцевого числа.

Зокрема, в цих позначеннях Фоннеймановська машина належить до *IssDss*.

Незважаючи на те, що *c* и *m* в принципі не мають певної верхньої межі, вони відображають різні властивості архітектури комп'ютера. Описувач з припускає жорсткі обмеження зверху з боку апаратури, і відповідний параметр не може бути значно збільшений відносно простими засобами. Прикладом може служити число інструкцій, упакованих в командному слові VLIW-комп'ютера. З іншого боку, описувач *m* використується тоді, коли позначається величина може бути легко змінена, тобто іншими словами, комп'ютер по даному параметру масштабується. Наприклад, відносна простота збільшення числа процесорних елементів у системі MPP є підставою для того, щоб віднести її до класу *IssDss*.

Звичайно ж, відмінність між *c* і *m* в достатній мірі умовне і, як правило, породжує масу питань. Наприклад, як описати машину, в якій процесори пов'язані через загальну шину? З одного боку, немає ніяких принципових обмежень на число підключаються процесорів. Однак кожний додатковий процесор збільшує завантаженість шини, і при досягненні деякого порога підключення нових процесорів безглуздо. За допомогою якого символу описати таку систему: *c* чи *m*? Ми залишаємо це питання відкритим.

На основі цих позначень можна виділити наступні класи комп'ютерів:

- *IssDss* – класичні машини Фоннеймановського типу;

– *IssDsc* – Фоннеймановська машини, в яких закладена можливість вибирати дані, розташовані з різним зсувом щодо одного і того ж адреси і над якими буде виконана одна і та ж операція. Прикладом можуть служити комп'ютери, що мають команди типу одночасного виконання двох операцій додавання над даними у форматі півслова, розташованими за вказаною адресою;

– *IssDsm – SIMD* – комп'ютери без можливості отримання унікальної адреси для даних у кожному процесорному елементі. Сюди входять, наприклад, MPP, Connection Machine 1 і систоличні масиви;

– *IssDmm – SIMD* – комп'ютери, що мають можливість незалежної модифікації адрес операндів у кожному процесорному елементі, наприклад, ILLIAC IV і Connection Machine 2;

– *IscDcc* – обчислювальні системи, що вибирають і виконують одночасно кілька команд, для доступу до яких використовується один адресу. Типовим прикладом є VLIW-комп'ютери;

– *ImmDmm* – до цього класу відносяться всі комп'ютери типу MIMD.

Досить ясно, що не потрібно розглядати всі можливі комбінації описувачів s, c і m, т. к. архітектура реальних комп'ютерів накладає ряд цілком розумних обмежень, зокрема і $w(Ia) \le w(Iv)$ *I* $w(Da) \le w(Dv)$.

Підводячи підсумок, можна відзначити два позитивних моменти у класифікації Шнайдера: більш виборча систематизація SIMD- комп'ютерів і можливість опису нетрадиційних архітектур типу систолических масивів або комп'ютерів з довгим командним словом. Разом з тим, майже всі обчислювальні системи типу MIMD знову потрапили в один і той же клас *ImmDmm*. Це означає, що критерій класифікації, заснований лише на потоках команд і даних без урахування розподіленості пам'яті і топології міжпроцессорной зв'язку, занадто слабкий для подібних систем.

Класифікація Д. Скіллікорна. У 1989 році була зроблена чергова спроба розширити класифікацію Флінна і тим самим подолати її недоліки. Д. Скіллікорн розробив підхід, придатний для опису властивостей багатопроцесорних систем і деяких нетрадиційних архітектур, зокрема, dataflow.

Пропонується розглядати архітектуру будь-якого комп'ютера, як абстрактну структуру, що складається з чотирьох компонентів:

– процесор команд (IP-Instruction Processor) – функціональне пристрій, що працює як інтерпретатор команд; в системі, взагалі кажучи, може бути відсутнім;

– процесор даних (DP-Data Processor) – функціональне пристрій, що працює як перетворювач даних відповідно з арифметичними операціями;

– ієрархія пам'яті (IM-Instruction memory, DM-Data memory) – запам'ятовуючий пристрій, в якому зберігаються дані і команди, що пересилаються між процесорами;

– перемикач – абстрактне пристрій, що забезпечує зв'язок між процесорами і пам'яттю.

Функції процесора команд багато в чому схожі з функціями пристроїв управління послідовних машин і, згідно Д. Скіллікорну, зводяться до наступних. На основі свого стану і отриманої від DP інформації IP виконує такі дії: визначає адресу команди, яка буде виконуватися наступної; здійснює доступ до IM для вибірки команди; отримує і декодує обрану команду; повідомляє DP команду, яку треба виконати; визначає адреси операндів і посилає їх в DP; отримує від DP інформацію про результат виконання команди.

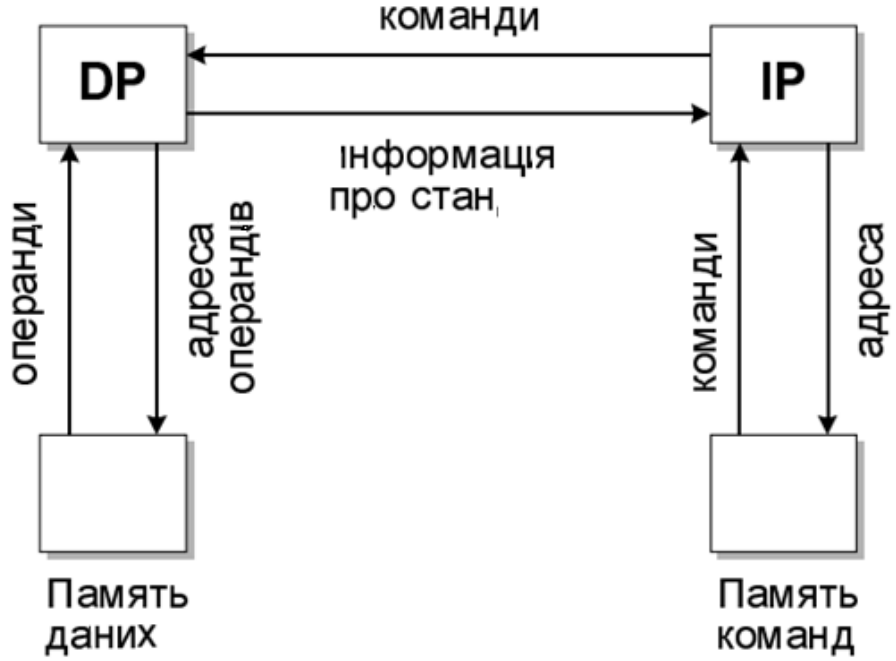


Рисунок 3.5. – Фонеймановська архітектура в термінах класифікації Д. Скіллікорна

Функції процесора даних роблять його багато в чому схожим на арифметичний пристрій традиційних процесорів. DP отримує від IP команду, яку треба виконати; отримує від IP адреси операндів; вибирає операнди із DM; запам’ятовує результат в DM; повертає в IP інформацію про стан після виконання команд.

Структура традиційної фон – неймановской архітектури в термінах таким чином певних основних частин комп'ютера показана на рис. 3.5. Це один з найпростіших видів архітектури, що не містять перемикачів. Для опису складних паралельних обчислювальних систем Д. Скіллікорн зафіксував чотири типи перемикачів без будь-якої явної зв'язку з типом пристроїв, які вони з'єднують:

– 1 – 1 – перемикач такого типу пов'язує пару функціональних устройств;

– *n*-*n* – перемикач пов'язує кожний пристрій з однієї множени пристроїв з відповідним йому пристроєм з іншої множини, тобто фіксує попарно зв'язок;

– *1-n* – перемикач з'єднує одне виділене пристрій з усіма функціональними пристроями з деякого набору;

– *n* x *n* – кожне функціональний пристрій одного безлічі може бути пов'язано з будь-яким пристроєм іншого безлічі, і навпаки.

Прикладів подібних перемикачів можна навести багато. Так, всі матричні процесори мають перемикач типу 1-*n* для зв'язку єдиного процесора команд з усіма процесорами даних. У комп'ютерах сімейства Connection Machine кожен процесор даних має свою локальну пам'ять, отже, такий зв'язок буде описуватися як *n* – *n*. У той же час, кожен процесор команд може зв'язатися з будь-яким іншим процесором, що відповідає опису *n*x*n*.

Класифікація Д. Скіллікорна будується на основі наступних восьми характеристик:

- кількість процесорів команд IP;
- число запам'ятовуючих пристроїв (модулів пам'яті) команд IM;
- тип перемикача між IP і IM;
- кількість процесорів даних DP;
- число запам'ятовуючих пристроїв (модулів пам'яті) даних DM;
- тип перемикача між DP і DM;
- тип перемикача між IP і DP;
- тип перемикача між DP і DP.

Розглянемо комп'ютер Connection Machine 2. У термінах даних характеристик його можна описати таким чином:

$$(1, 1, 1 – 1, n, n, n – n, 1 – n, n \text{x} n).$$

Умовне зображення його архітектури наведено на рис. 3.6.

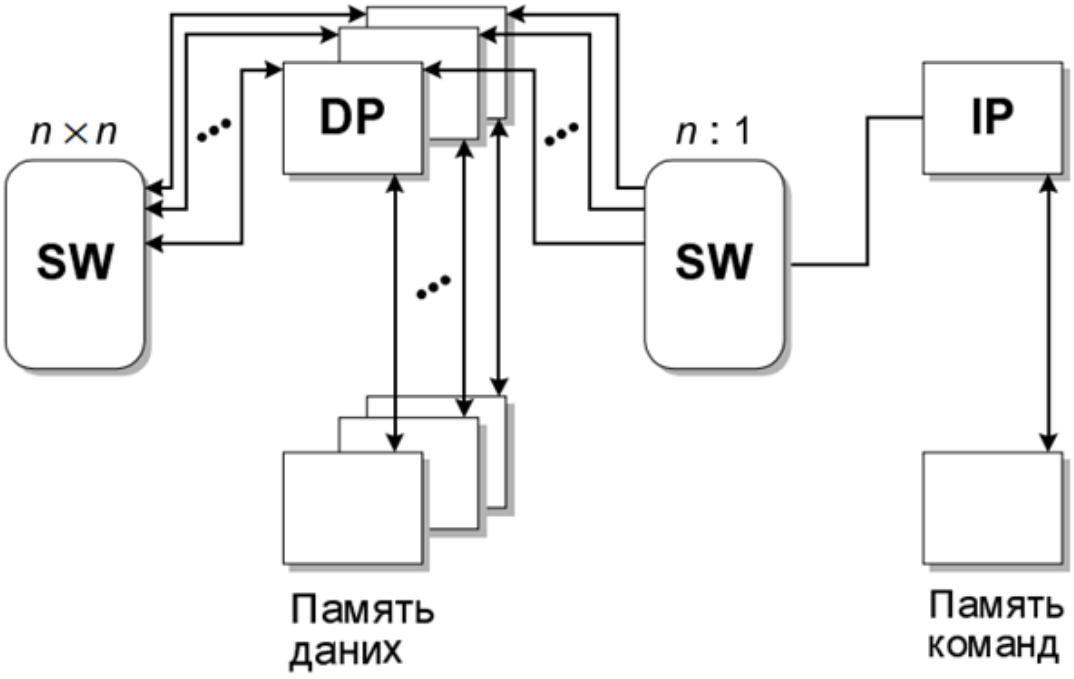


Рисунок 3.6. – Архітектура Connection Machine 2 в класифікації Д. Скіллікорна

Для сильно пов'язаних мультипроцессорів типу BBN Butterfly ситуація інша. Такі системи складаються з безлічі процесорів, з'єднаних з модулями пам'яті за допомогою перемикача. Затримка при доступі будь-якого процесора до будь-якого модулю пам'яті приблизно однакова. Зв'язок і синхронізація між процесорами здійснюється через загальні (колективні) змінні. Опис таких машин в рамках даної класифікації виглядає так:

(*n, n, n – n, n, n, n x n, n – n,* немає).

Саму архітектуру можна зобразити так, як показано на рис. 3.7.

Рисунок 3.7. – Архітектура BBN Butterfly в класифікації Д.Скіллікорна

Використовуючи введені характеристики і припускаючи, що розгляд кількісних характеристик можна обмежити тільки трьома можливими варіантами значень: 0, 1 і *n* (тобто більше одиниці), можна отримати різні класи архітектур.

Цікаво, що серед сформульованих Скіллікорном цілей, яким повинна служити добре побудована класифікація архітектур, є й така: "класифікація повинна показувати, за рахунок яких структурних особливостей досягається збільшення продуктивності різних обчислювальних систем". Ця мета дуже співзвучна цілям наших досліджень. Коли користувач приходить на нову обчислювальну систему, то, як правило, йому відома лише її пікова продуктивність. Однак цього явно недостатньо. Якщо йому дійсно потрібна висока продуктивність, то, хоче він цього чи ні, йому доведеться мати справу з цілим спектром суміжних питань. Практично всі вони пов'язані з особливостями архітектури комп'ютера. Ось тут-то і стала в нагоді б гарна класифікація.

Попадання комп'ютера в той чи інший клас відразу давало б користувачеві інформацію про особливості його програмування, методах досягнення високої продуктивності, причини низької продуктивності. Опосередковано таку інформацію отримати можна, але тільки в загальних рисах. Зокрема, всі комп'ютери в першому наближенні можна поділити на комп'ютери із загальною і розподіленою пам'яттю. Для комп'ютерів із загальною пам'яттю користувачеві не потрібно піклуватися про розподіл даних, а для програмування можна використовувати просту технологію OpenMP. На комп'ютерах з розподіленою пам'яттю йому, швидше за все, доведеться працювати в термінах MP1 і самому піклуватися про розподіл і пересиланнях даних. У загальних рисах все вірно, але, чесно кажучи, тільки в найзагальніших. У такому описі опущено занадто багато важливих деталей, без яких складання ефективних паралельних програм просто неможливо. Раз ці деталі істотні, то вони повинні бути відображені в класифікації. Комп'ютерів багато, деталей ще більше, звідси і безліч класифікацій, і інтерес до даної проблеми в цілому.

3.3. Паралельні комп'ютери із загальною пам'яттю

До комп'ютерів із загальною пам'яттю у користувачів завжди було неоднозначне ставлення. З одного боку, програмування комп'ютерів цього класу значно простіше, ніж, скажімо, програмування обчислювальних кластерів з розподіленою пам'яттю. Справді, не потрібно думати про розподіл масивів, внутрішній паралелізм програм описується просто, процес налагодження йде швидше. З іншого боку, класичні представники цього класу мають і два недоліки: невелике число процесорів і дуже високу вартість.

Щоб збільшити число процесорів, але зберегти можливість роботи в рамках єдиного адресного простору, пропонувалися різні рішення (див. § 2.2). Одним з поширених варіантів є рішення на основі архітектури ccNUMA (cache coherent Nom Uniform Memory Access). У такій архітектурі пам'ять всього комп'ютера фізично розподілена, що значно збільшує потенціал його масштабованості. Разом з тим, пам'ять логічно залишається спільною. Це дає можливість використання всіх технологій і методів програмування, створених для SMP – комп'ютерів. Додатково в такій архітектурі вміст кеш -пам'яті окремих процесорів на рівні апаратури узгоджується з вмістом оперативної пам'яті (вирішується проблема когерентності кешем, cache coherence problem). Значно збільшуючи число процесорів в порівнянні з SMP – комп'ютерами, архітектура ccNUMA привносить додаткову особливість, невластиву комп'ютерам із загальною пам'яттю. Час звернення до пам'яті залежить від того, чи є це зверненням до локальної або віддаленої пам'яті. Процес написання програм залишається колишнім, і фізична розподіленість пам'яті програмісту не видно. Однак ясно, що ефективність програм напряму залежить від ступеня "неоднорідності" доступу до пам'яті.

Проведемо дослідження архітектури комп'ютерів даного класу на прикладі обчислювальної системи Hewlett-Packard Superdome. Комп'ютер

з'явився в 2000 році, а в листопадовій редакції списку Top500 2001 їм вже були зайняті 147 позицій.

Комп'ютер HP Superdome в стандартній комплектації може об'єднувати від 2 до 64 процесорів з можливістю подальшого розширення системи. Всі процесори мають доступ до загальної пам'яті, організованої в відповідності з архітектурою ccNUMA. Це означає, що, по-перше, всі процеси можуть працювати в єдиному адресному просторі, адресуючи будь-який байт пам'яті за допомогою звичайних операцій читання / запису. По-друге, доступ до локальної пам'яті в системі буде йти трохи швидше, ніж доступ до віддаленої пам'яті. По-третє, проблеми можливої не відповідності даних, викликані кеш-пам'яттю процесорів, вирішені на рівні апаратури.

У максимальній конфігурації Superdome може містити до 256 Гбайт оперативної пам'яті. Найближчі плани компанії – реалізувати можливість нарощування пам'яті комп'ютера до 1 Тбайта.

Архітектура комп'ютера спроектована таким чином, що в ній можуть використовуватися кілька типів мікропроцесорів. Це, звичайно ж, традиційні для обчислювальних систем Hewlett-Packard процесори сімейства PA: PA-8600 і PA -8700. Разом з тим, система повністю підготовлена і до використання процесорів наступного покоління з архітектурою IA -64, розробленої спільно компаніями HP Superdome. При заміні існуючих процесорів на процесори 1A -64 гарантується двійкова сумісність додатків на системному рівні. Надалі, якщо іншого не обумовлено, будемо розглядати конфігурації HP Superdome на базі процесора PA -8700.

Основу архітектури комп'ютера HP Superdome складають обчислювальні комірки (cell), пов'язані ієрархічною системою перемикачів. Кожна комірка є симетричним мультипроцесором, реалізованих на одній платі, в якому є всі необхідні компоненти (рис. 3.8):

- процесори (до 4 -х);
- оперативна пам'ять (до 16 Гбайт);
- контролер комірки;
- перетворювачі харчування;
- зв'язок з підсистемою введення / виводу (опціонально).

Цікаво, що осередки Superdome багато в чому схожі на аналогічні архітектурні елементи інших сучасних ccNUMA комп'ютерів. У Superdome таким елементом є осередок, в сімействі SGI Origin 3x00 це вузол (node), а в комп'ютерах серії Compaq AlphaSever GS320 – QBB (Quad Building Block). У всіх системах в кожному елементі міститься по чотири процесори.

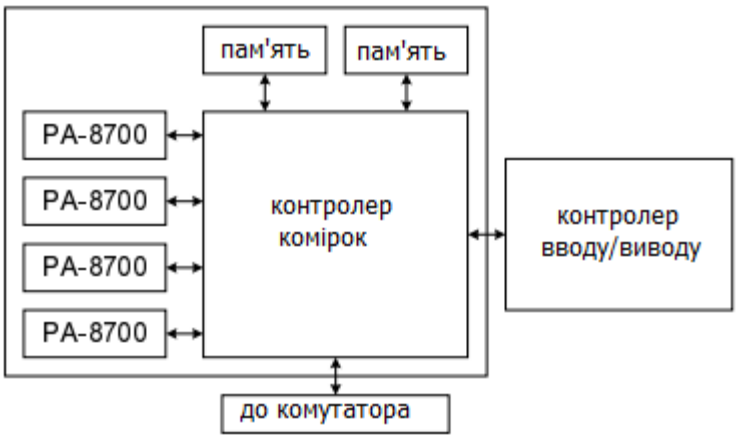


Рисунок 3.8. – Структура комірки комп’ютера HP Superdome

Центральне місце в архітектурі осередку Superdome займає контролер комірки. Незважаючи на настільки буденне назва, контролер – це складне пристрій, що складається з 24 мільйонів транзисторів. Для кожного процесора осередку є власний порт в контролері. Обмін даними між кожним процесором і контроллером йде зі швидкістю 2 Гбайт / с.

Пам'ять осередку має ємність від 2 до 16 Гбайт. Конструктивно вона розділена на два банки, кожен з яких має свій порт в контролері осередки. Швидкість обміну даними між контролером і кожним банком становить 2 Гбайт / с, що дає сумарну пропускну здатність тракту контролер – пам'ять 4 Гбайт / с.

З'єднання контролера комірки з контролером пристроїв введення / виводу (12 слотів PCI) встановлюється опціонально.

Один порт контролера комірки завжди пов'язаний із зовнішнім комутатором. Він призначений для обміну процесорів осередки з іншим процесорами системи. Швидкість роботи цього порту дорівнює 8 Гбайт / с.

Виконуючи інтерфейсні функції між процесорами, пам'яттю, іншими осередками і зовнішнім світом, контролер комірки відповідає і за когерентність кеш -пам'яті процесорів.

- контролер комірки;
- перетворювачі живлення;
- зв'язок з підсистемою введення/виведення.

Цікаво, що комірки Superdome багато в чому схожі на аналогічні архітектурні елементи других сучасних ccNUMA комп'ютерів. У Superdome таким елементом є комірка, в сімействі SGI Origin 3x00 це вузол (node), а в комп'ютерах Серії Compag AlphaServer GS320-QBB (Quad Building Block). В усіх системах у кожному елементі міститься по чотири процесори.

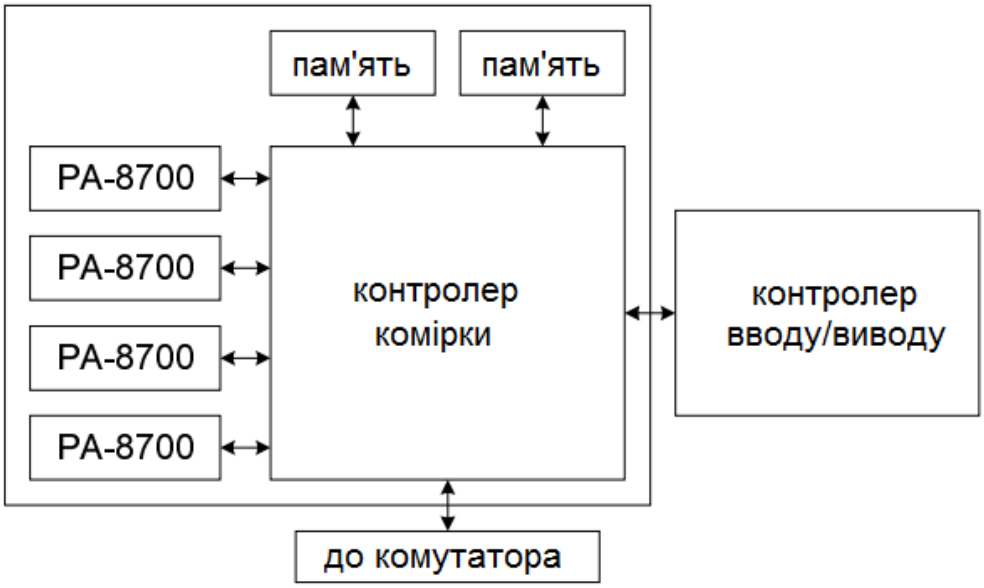


Рисунок 3.9. – Структура комірки комп’ютера HP Superdome

Центральне місце в архітектурі комірки Superdome займає контролер комірки. Незважаючи на таку буденну назву, контролер – це складний пристрій,що складається з 24 мільйонів транзисторів. Для кожного процессора комірки є власний порт в контролері. Обмін даними між кожним процессором і контролером йде з швидкістю 2 Гбайт/с.

Пам'ять комірки має місткість від 2 до 16 Гбайт. Конструктивно вона розділена на два банки, кожен зяких має свій порт в контролері комірки. Швидкість обміну даними між контролером і кожним банком складає 2 Гбайт/с, що дає сумарну пропускну здатність тракту контроллер-память 4 Гбайт/с.

З'єднання контролера комірки з контролером пристроїв введення/виведення (12 слотів PCI) встановлюється опціонально.

Один порт контроллера комірки завжди пов'язаний із зовнішнім комутатором. Він призначений для обміну процесорів комірки з іншим процесорами системи. Швидкість роботи цього порту дорівнює 8 Гбайт/с.

Виконуючи інтерфейсні функції між процесорами, пам'яттю, іншими комірками і зовнішнім світом, контроллер комірки відповідає і за когерентність кеш-пам'яті процесорів.

Комірка – це базовий чотирьохпроцесорний блок комп'ютера. У 64 – процесорній конфігурації Superdome складається з двох стійок, в кожній стійці по 32 процесора (Рис. 3.10).

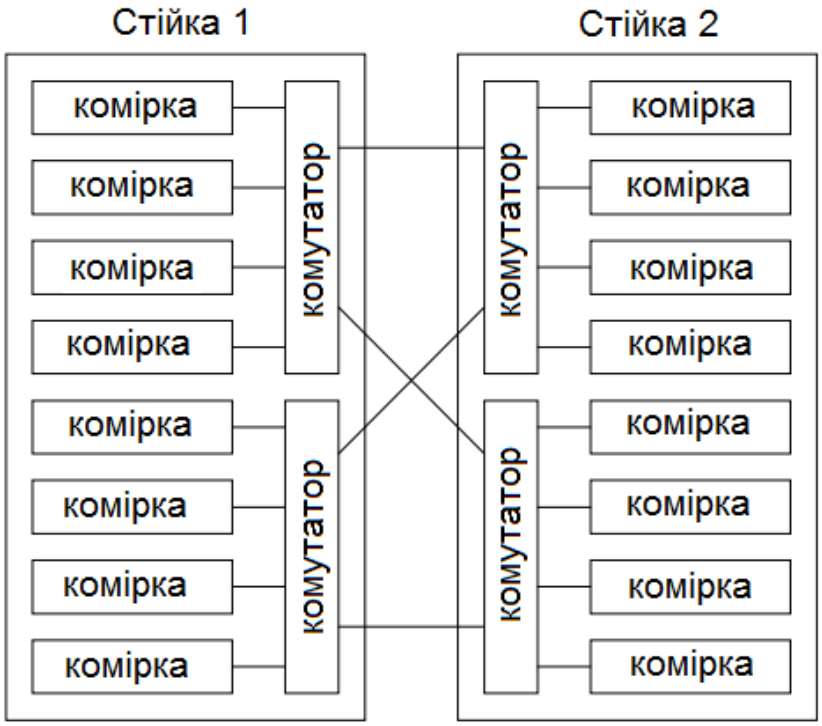


Рисунок 3.10. – Загальна структура комп’ютера HP Superdome

Кожна стійка містить по два восьмипортових не блокуючих комутатора. Усі порти комутаторів працюють із швидкістю 8 Гбайт/с. До кожного комутатора підключаються чотири комірки. Три порти коммутатора задіяні для зв'язку з іншими коммутаторами системи (один в цій же стійці, і два комутатори в іншій). Порт, що залишився, зарезервований для зв'язку з іншими системами HP Superdome, що дає потенційну можливість для формування багатовузлової конфігурації комп'ютера із загальним числом процесорів більше 64.

Одним з центральних питань будь-якої обчислювальної системи з архитектурою ccNUMA є різниця в часі при зверненні процесора до локальних і видалених елементів пам'яті. В ідеалі хотілося б, щоб цієї різниці, як в SMP-комп’ютері, не було зовсім. Проте у такому разі система свідомо буде погано масштабована. У комп'ютері HP Superdome можливі три види затримок при зверненні процесора до пам'яті, які являються свого роду платою за високу масштабованість системи в цілому:

– процесор і пам'ять розташовуються в одній комірці; в цьому випадку затримка мінімальна;

– процесор і пам'ять розташовуються в різних комірках, але обидві ці комірки приєднані до одного і того ж комутатора;

– процесор і пам'ять розташовуються в різних комірках, причому обидві комірки приєднані до різних комутаторів; в цьому випадку запит повинен пройти через два комутатори і тоді затримки будуть максимальними.

Ясно, що величина затримки залежить не лише від взаємного розташування процесора і пам'яті, але і від числа процесорів. Не менш важливим параметром є і характер обчислювального навантаження. Наприклад,

затримка може змінюватися залежно від числа одночасно працюючих додатків. У таблиці 3.1 показана залежність затримки від числа процесорів в двох характерних ситуаціях. У першому випадку працюють однонитьові додатки, а в другому – багатонитяна програма. На відміну від першого випадку, в другому випадку з'являються додаткові витрати, необхідні для підтримки когерентності кеш-пам'яті процесорів. В обох випадках наводяться усереднені показники затримки. Передбачається, що запити до пам'яті розподілені рівномірно.

Таблиця 3.1.

Затримка в залежності від кількості процесорів і завантаження

Число процесорів	**Однонитні програми Нс**	**Багатонитна програма Нс**
4	174	235
8	208	266
16	228	296
32	261	336
64	275	360

Для розглянутих видів навантаження, при зроблених вище припущеннях про розподіл запитів до пам'яті, показані результати є дуже непоганими. Насправді, коефіцієнт збільшення затримки при переході від мінімальної 4-процесорної конфігурації до 64-процесорної складає лише 1,6 разів. Це означає, що у багатьох випадках користувач має право сподіватися на ефективну реалізацію своїх програм, створених їм для традиційних SMP-комп’ютерів.

Комп'ютер HP Superdome має масу цікавих особливостей. Частково, програмно-апаратне середовище комп'ютера дозволяє його настроїти різним способом. Superdome може бути класичним єдиним комп’ютером із загальною пам'яттю. Проте його можна конфігурувати і таким чином, що він буде сукупністю незалежних розділів (nPartitions), працюючих під різними операційними системами, зокрема, під HP UX, Linux і Windows 2000. Організація ефективної роботи з великим числом зовнішніх пристроїв, можливості "гарячої" заміни усіх основних компонентів апаратури, резервування, моніторинг базових параметрів – усе це залишиться за рамками нашого обговорення.

Вивчивши особливості архітектури HP Superdome вцілому, коротко розглянемо структуру використовуваного в ньому процесора PA- 8700. Якщо ми хочемо ефективно використати усю систему, необхідно, хоч би в загальних рисах, мати уявлення про особливості її базової компоненти.

Процесор має тактову частоту 750 Мгц. Працюючи з максимальним завантаженням, він може виконувати чотири арифметичні операції за такт. Ці два параметри визначають значення його пікової продуктивності – 3Гфлопс. Звідси виходить і значення пікової продуктивності базової 64-процесорної системи HP Superdome – 192 Гфлопс.

Процесор PA-8700 має супер скалярну архітектуру. На кожному такті він виконує стільки операцій, скільки (1) дозволяє інформаційна структура коду і (2) скільки в даний момент є доступних функціональних пристроїв. Всього процесор PA- 8700 містить 10 функціональних пристроїв: чотири пристрої для цілочисельної арифметики і логіки, чотири пристрої для роботи з речовою арифметикою і два пристрої для операцій читання/запису. На кожному такті облаштування вибірки команд може прочитувати 4 команди з кеш-пам'яті команд.

Починаючи з PA- 8500, процесори цього сімейства мають велику кэш-память першого рівня LI, реалізовану прямо на кристалі. У PA- 8700 об'єм кеш-пам'яті дорівнює 2,25 Мбайт, з яких 1,5 Мбайт відводиться під кеш даних, а решту 0,75 Мбайт, – це кеш команд. Уся кеш-пам'ять процесора є асоціативною каналами (4-way set associative cache).

В цілому область використання комп’ютерів HP Superdome виключно широке. Досить сказати, що дві великі інсталяції цього комп’ютера в Росії відповідають різним завданням і областям. Один 64-процесорний комп’ютер HP Superdome встановлений в міжвідомчому суперкомп’ютерному центрі для вирішення широкого спектру науково-технічних завдань, а 72-процесорна конфігурація працює в Ощадбанку Росії. На закінчення, як і в попередньому параграфі, виділимо ті особливості обчислювальних систем із загальною пам’яттю, які знижують їх продуктивність на реальних програмах. Не повинне складатися враження, що кожен параграф цієї глави ми хочемо закінчити на песимістичній ноті. Проте сприймати реальність треба з розплющеними очима, чітко представляючи не лише переваги використання паралельних комп’ютерів, але і підводні камені, що зустрічаються на цьому шляху.

Закон Амдала носить універсальний характер, тому він згадується разом з усіма паралельними системами. Не є виключенням і комп’ютери з загальною пам’яттю. Якщо в програмі 20 % всіх операцій повинні виконуватися строго послідовно, то прискорення більше 5 отримати не можна незалежно від числа використаних процесорів (вплив кеш пам’яті зараз не розглядається). Це треба врахувати і перед адаптацією старої послідовної програми до такої архітектури, і в процесі проектування нового паралельного коду.

Для комп’ютерів із загальною пам’яттю додатково слід врахувати і такі міркування. Наявність фізичної загальної пам’яті стимулює до використання моделей паралельних програм також із загальною пам’яттю. Це цілком природно і виправдано. Проте в цьому випадку виникають додаткові ділянки послідовного коду, пов’язані з синхронізацією доступу до загальних даних, наприклад, критичні секції. Відносно подібних конструкцій описів відповідної технології програмування може і не бути ніякого застереження, проте реально ці фрагменти можуть бути послідовними ділянками.

Робота з пам’яттю є дуже тонким місцем в системах цього класу. Одну з причин зниження продуктивності – неоднорідність доступу до пам’яті, ми вже обговорювали. Мірами однорідності на рівні 5-10 % серйозних проблем не створить. Проте різниця в часі доступу до локальної і віддаленої пам’яті в декілька разів вимагає від користувача дуже акуратного програмування. В

цьому випадку йому доведеться вирішувати питання, аналогічні розподілу даних для систем з розподіленою пам'яттю. Іншу причину – конфлікти при зверненні до пам'яті – ми детально не розбирали, але вона також характерна для багатьох SMP–систем. Наявність кеш-пам'яті у кожного процесора теж приносить свої додаткові особливості. Найбільш суттєва з них полягає в необхідності забезпечення узгодженості вмісту кеш пам'яті. Звідси з'явилися і перші дві букви в абревіатурі ccNUMA. Чим рідше залучається апаратура у вирішенні цієї проблеми, тим менше накладних витрат супроводжує виконання програми. З цієї ж причини у багатьох системах із загальною пам'яттю існує режим виконання паралельної програми з прив'язкою процесів до процесорів.

Збалансованість обчислювльного навантаження також характерна для паралельних систем, як і закон Амдала. У разі систем із загальною пам'яттю ситуація спрощується тим, що практично завжди системи є однорідними. Вони містять однакові процесори, тому про складну стратегію розподілу роботи, як правило, не йдеться.

Будь-який сучасний процесор має складну архітектуру, що об'єднує і декілька рівнів пам'яті, і безліч функціональних пристроїв. Реальна продуктивність окремого процесора може відрізнятися від його ж пікової в десятки разів. Чим вище міра використання можливостей кожного процесора, тим вище загальна продуктивність обчислювальної системи.

І знову, як і в попередньому параграфі, перерахування особливостей комп'ютера, що впливають на його продуктивність, можна продовжувати і далі. Тут вони свої, але знову-таки всі вони в тій чи іншій мірі проявляються в кожній програмі. Звідси і проблеми з низькою продуктивністю, звідси і потенційні проблеми і у користуввачів. Проблеми вирішувані, але їх треба ясно уявляти, щоб вибрати правильний спосіб рішення.

3.4. Обчислювальні системи з розподіленою пам'яттю

Ідея побудови обчислювальних систем цього класу дуже проста. Береться якась кількість обчислювальних вузлів, які обєднуються один з одним деяким комунікаційним серидовищем. Кожен обчислювальний вузол має один або декілька процесорів і свою власну локальну пам'ять, що розділяється цими процесорами. Розподіленість в пам'яті означає те, що кожен процесор має безпосередній доступ тільки до локальної пам'яті свого вузла. Доступ до даних, розташованих в пам'яті інших вузлів, виконується довше і іншими, складнішими способами. Останнім часом в якості вузлів все частіше і частіше використовують повнофункціональні комп'ютери, що містять, наприклад, і власні зовнішні пристрої. Комунікаційне середовище може спеціально проектуватися для даної обчислювальної системи або бути стандартною мережею, доступною на ринку.

Переваг у такої системи організації перелельних комп'ютерів багато. Зокрема, покупець може досить точно підібрати конфігурацію залежно від наявного бюджету і своїх потреб в обчислювальній потужності. Співвідношення ціна/продуктивність систем з розподіленою пам'яттю нижче,

ніж у комп'ютерів інших класів. І головне, така схема дає можливість практично необмежено нарощувати число процесорів в системі і збільшувати її продуктивнсть. Велике число процесорів, що підключаються, навіть визначило спеціальну назву для систем цього класу: комп'ютери з масовим паралелізмом або масивно-паралельні комп'ютери. Вже зараз у світі існують десятки комп'ютерів, у складі яких працює більше тисячі процесорів.

Широке поширення комп'ютери з такою архітектурою отримали з початку 90-х років минулого століття. Серед них можна назвати Intel Paragon, IBM SP1/SP2, Cray T3D/T3E і ряд інших. За великим рахунком, відмінність між ними полягає у використовуваних процесорах і організації комунікаційного середовища. Комп'ютер Intel Paragon був побудований на основі процесорів Intel i860, розташованих у вузлах прямокутної двовимірної решітки. В обчислювальних вузлах IBM SP в міру розвитку цього сімейства використовувалося декілька процесорів, зокрема, PowerPC, P2SC, Power3. Їх взаємодія йде через ієрархічну систем високопродуктивних комутаторів, що дає потенційну можливість спілкування кожного вузла з кожним. Сімейство комп'ютерів Cray T3D/T3E спирається на процесори DEC Alpha і топологію трьохвимірного тора.

Цей параграф ми розпочнемо з опису архітектури комп'ютерів сімейства Cray T3D/T3E. Будучи представниками обговорюваного класу обчислювальних систем, ці комп'ютери мають ряд цікавих додаткових особливостей, що і зумовило наш вибір.

Отже, комп'ютери Cray T3D/T3E – це масивно-паралельні комп'ютери з розподіленою пам'яттю, що об'єднують в максимальні конфігураціїї більше 2000 процесорів. Як і будь-які комп'ютери цього класу, вони містять дві основні компоненти: вузли і комунікаційне середовище.

Всі вузли комп'ютера діляться на 3 групи. Коли користувач підключається до комп'ютера, то він потрапляє на вузли, що управляють. Ці вузли працюють в розрахованому на багато користувачів режимі, на цих же вузлах виконуються однопроцесорні програми і працюють командні файли. Вузли операційної системи не доступні користувачам безпосередньо. Ці вузли підтримують виконання багатьох системних сервісних функцій ОС, зокрема, роботу з файловою системою. Обчислювальні вузли комп'ютера призначені для виконання програм користувача в монопольному режимі. При запуску програми виділяється необхідне число вузлів, які за нею закріплюються аж до моменту її завершення. Гарантується, що ніяка програма не зможе зайняти обчислювальні вузли, на яких вже працює інша програма. Число вузлів кожного типу залежить від конфігурації системи. Зокрема, дані, взяті з двох реальних конфігурацій Cray T3E, виглядають так: 24/16/576 або 7/5/260 (управляючі вузли/вузли ОС/обчислювальні вузли).

Кожен вузол останніх моделей складається з процесорного елемента (ПЕ) і мережевого інтерфейсу. Процесорний елемент містить один процесор Alpha, локальну пам'ять і допоміжні підсистеми. У моделі Cray T3E-1200E використовувалися мікропроцесори DEC Alpha 21164/600 МГц. В моделі Cray T3E-1350 у вузлах використовуються мікропроцесори Alpha 21164A (EV5.6) з

тактовою частотою 675 МГц і піковою продуктивністю 1,35 Гфлопс. Цікаво, що комп'ютері серії Cray T3D містили в кожному вузлі по два незалежних процесори DEC Alpha 211064/150 МГц.

Локальна пам'ять кожного процесорного елементу є частиною фізично розподіленої, але логічно розділяючої пам'яті всього ком'пютера. Пам'ять фізично розподілена, так як кожен ПЕ містить свою локальну пам'ять. В той же час пам'ять розділяється всіма ПЕ, оскільки будь-який ПЕ через свій мережевий інтерфейс може звертатися до пам'яті будь-якого іншого ПЕ, не перериваючи його роботу.

Мережевий інтерфейс вузла пов'язаний з відповідним мережевим маршрутизатором, який є частиною комунікаційної мережі. Всі маршрутизатори розташовані по вузлах трьохвимірних цілочисельних прямокутних решіток і сполучені між собою відповідно до топології трьохвимірного тора (Рис. 3.11). Це означає, що кожен вузол має 6 безпосередніх сусідів незалежно від того, де він розташований: в середині паралелепіпеда, на ребрі, на грані, або в його вершині.

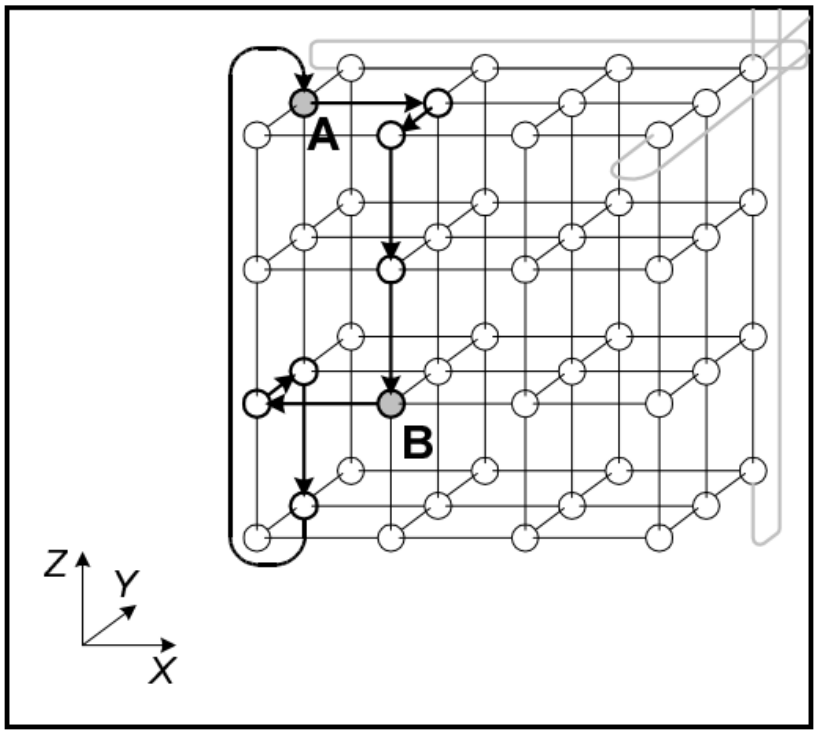


Рисунок 3.11. – Комунікаційна решітка комп'ютера Cray T3E

Подібна організація комунікаційної мережі має багато переваг, серед яких відзначимо дві. По-перше, це можливість вибору альтернативного маршруту для обходу пошкоджених зв'язків. А по-друге, швидкий зв'язок граничних вузлів і невелике середнє число переміщень по тору при взаємодії різних ПЕ.

Кожен елементарний зв'язок між двома вузлами – це два однонаправлених канали передачі даних, що допускають одночасний обмін даними в протилежних напрямках. У моделі Cray T3E-1200E максимальна

швидкість передачі даних між вузлами дорівнює 480 Мбайт/с, латентність на рівні апаратури становить менше 1 мкс.

Комунікаційна мережа утворює трьохвимірну решітку, з'єднуючи мережеві маршрутизатори вузлів у напрямках X, Y, Z. Вибір маршруту для обміну даними між двома вузлами A і B відбувається наступним чином. Відштовхуючись від вершини A, мережеві маршрутизатори спочатку виконують зсув по вимірюванню X доти, поки координата чергового транзитного вузла і вершини B по вимірюванню X не стануть рівними. Потім аналогічна процедура виконується по Y а в кінці по Z (див. Рис. 3.14). Так як зсув може бути як позитивним, так і негативним, то цей механізм допомагає мінімізувати число переміщень по мережі і обійти пошкоджені зв'язки. Мережеві маршрутизатори спроектовані таким чином, щоб здійснювати паралельний транзит даних по кожному з трьох вимірів X, Y і Z одночасно.

Безумовно, цікавою особливістю архітектури комп'ютера є апаратна підтримка *бар'єрної синхронізації*. Бар'єр – це точка в програмі, при досягненні якої кожен процес повинен чекати до тих пір, поки інші процеси також не дійдуть до бар'єру. Лише після цієї події всі процеси можуть продовжувати роботу далі. Даний вид синхронізації часто використовується в програмах, але його реалізація засобами системи програмування супроводжується великими накладними витратами. Підтримка бар'єрів в апаратурі дозволяє звести ці витрати до мінімуму.

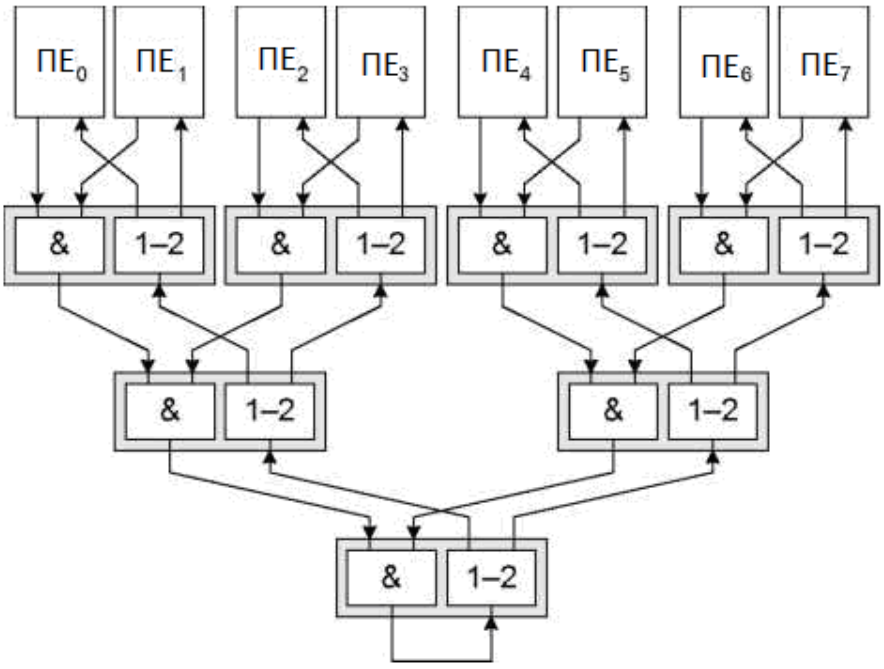


Рисунок 3.12. – Бар'єрна синхронізація в комп'ютерах Cray T3D/T3E

У схемах підтримки кожного ПЕ передбачено кілька вхідних і вихідних регістрів синхронізації. Кожен розряд цих регістрів з'єднаний зі своїм незалежним ланцюгом реалізації бар'єру. Всі ланцюги синхронізації однакові, а їх загальне число залежить від конфігурації комп'ютера. Кожен ланцюг будується за принципом двійкового дерева на основі двох типів пристроїв

(Рис. 3.12). Одні пристрої реалізують логічне множення ("&"), а інші виконують дублювання входу на два своїх виходи ("1-2"). Вихід останнього пристрою "&" є входом першого пристрою дублювання "1-2". Пристрої дублювання за своїм ланцюгом поширюють отримані значення по всім ПЕ, записуючи їх у відповідний розряд вихідного регістру.

Розглянемо роботу одного ланцюга. У початковому стані відповідний розряд вхідного і вихідного регістра кожного ПЕ дорівнює нулю. Поява одиниці в вихідному розряді ПЕ буде сигналізувати про досягнення бар'єра всіма процесами. Як тільки процес доходить до бар'єру, то розряд вхідного регістра відповідного ПЕ перевизначається в одиницю. На виході будь-якого пристрою "&" одиниця з'явиться тільки в тому випадку, якщо обидва входи містять одиницю. Якщо який-небудь процес ще не дійшов до бар'єру, то нуль на вході цього ПЕ пройде по ланцюжку пристроїв логічного множення і визначить нуль на виході останнього пристрою "&". Отже, нулі будуть і у вихідних розрядах кожного ПЕ. Як тільки останній процес записав одиницю в свій вхідний розряд, одиниця появляється на виході останнього пристрою "&" і розноситься ланцюгом пристроїв дублювання по вихідних розрядами на кожному ПЕ. За значенням вихідного розряду кожен процес дізнається, що всі інші процеси дійшли до точки бар'єрної синхронізації.

Ці ж ланцюги в комп'ютерах даного сімейства використовуються і по-іншому. Якщо всі пристрої логічного множення у схемі замінити пристроями логічного складання, то вийде ланцюг для реалізації механізму "Еврика". На виході будь-якого пристрою логічного складання одиниця з'явиться в тому випадку, якщо одиниця є хоча б на одному його вході. Це значить, що як тільки один ПЕ записав одиницю у вхідний регістр, ця одиниця поширюється всім ПЕ, сигналізуючи про деяку подію на вихідному ПЕ. Найочевидніша область застосування даного механізму – це задачі пошуку.

Крім традиційних суперкомп'ютерів типу Cray T3E або IBM SP, клас комп'ютерів з розподіленою пам'яттю останнім часом активно расширюється за рахунок обчислювальних кластерів. Відразу обмовимося, що в комп'ютерній літературі поняття "кластер" вживається в різних значення. Зокрема, "кластерна" технологія використовується для підвищення швидкості роботи та надійності серверів баз даних або WEB-серверів. Тут ми будемо говорити тільки про кластери, орієнтовані на вирішення завдань обчислювального характеру.

Класичні суперкомп'ютери завжди асоціювалися з чимось більшим: величезні розміри, гігантська продуктивність, велика пам'ять і, звичайно ж, колосальна вартість. Сама по собі висока вартість здивування не викликає. Унікальні рішення, та ще й з рекордними характеристиками, не можуть бути дешевими. Однак прогрес в електроніці вніс серйозні корективи. У середині 90-х років минулого століття на ринку з'явилися недорогі та ефективні мікропроцесори і комунікаційні рішення. З'явилася реальна можливість створювати установки "Суперкомкомп'ютерного" класу із складових частин масового виробництва. Це і передбачило появу кластерних обчислювальних

систем, що є окремим напрямком розвитку комп'ютерів з масовим паралелізмом.

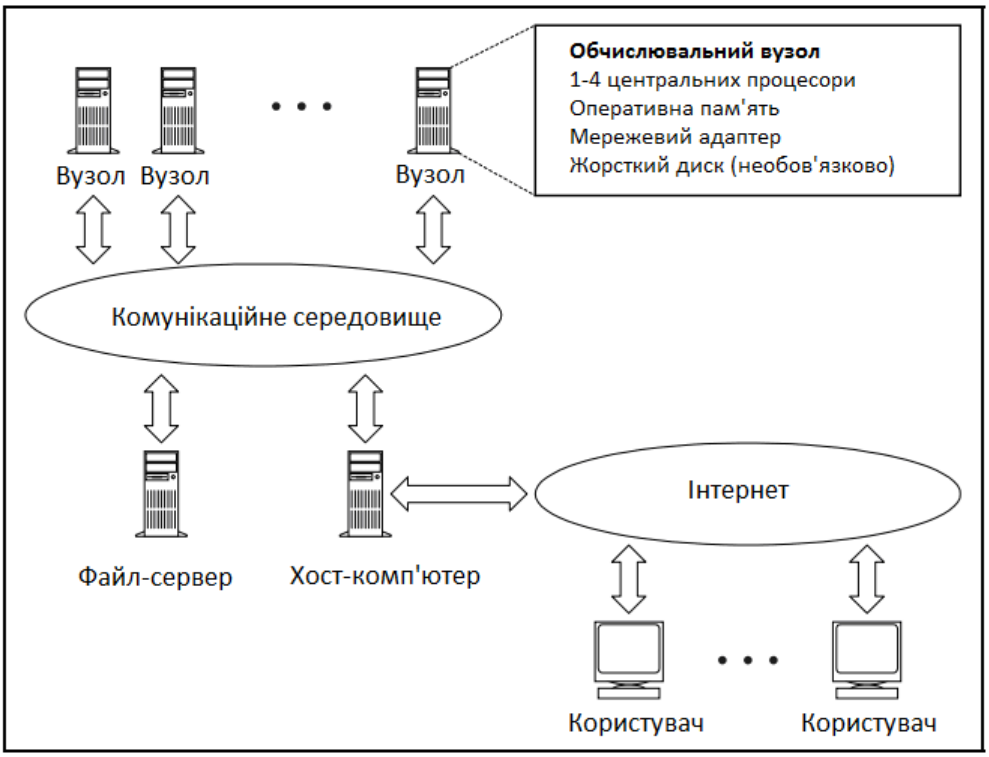


Рисунок 3.13. – Загальна схема обчислювального кластера

Якщо говорити коротко, то обчислювальний кластер є сукупністю комп'ютерів, об'єднаних в рамках деякої мережі для вирішення однієї задачі (Рис. 3.13). В якості обчислювальних вузлів зазвичай використовуються доступні на ринку однопроцесорні комп'ютери, двох- або чотирьох- процесорні SMP-сервери. Кожен вузол працює під керуванням своєї копії операційної системи, у якості якої найчастіше використовуються стандартні ОС: Linux, Windows NT. Solaris. Склад і потужність вузлів можуть мінятися навіть в рамках одного кластера, що дає можливість створювати неоднорідні системи. Вибір конкретного комунікаційного середовища визначається багатьма факторами: особливостями класу вирішуваних завдань, доступним фінансуванням, необхідністю подальшого розширення кластера і т. д. Можливо включення в конфігурацію кластеру спеціалізованих комп'ютерів, наприклад, файл-сервера. Як правило, представляється можливість віддаленого доступу на кластер через Інтернет.

Ясно, що простір для творчості при проектуванні кластерів величезний. Вузли можуть не містити локальних дисків, комунікаційне середовище може одночасно використовувати різні мережеві технології, вузли не повинні бути однаковими і т. д. Розглядаючи крайні точки, кластером можна вважати як пару ПК, пов'язаних локальною 10-мегабітною Ethernet-мережею, так і обчислювальну систему, створювану в рамках проекту Cplant в Національній лабораторії Sandia: 1400 робочих станцій на базі процесорів Alpha об'єднані високошвидкісною мережею Myrinet. Щоб краще відчути масштаб і технологію

існуючих систем, зробимо короткий огляд найбільш цікавих сучасних кластерних установок.

3.5 Кластерні проекти

Один з перших проектів, що дав ім'я цілому класу паралельних систем – Beowulf-кластери, виник у центрі NASA Goddard Space Flight Center (GSFC). Проект Beowulf стартував влітку 1994 року, і незабаром був зібраний 16-процесорний кластер на процесорах Intel 486DX4/100 МГц. На кожному вузлі було встановлено по 16 Мбайт оперативної пам'яті і по 3 мережевих карти для звичайної мережі Ethernet. Для роботи у такій конфігурації були розроблені спеціальні драйвери, що розподіляють трафік між доступними мережевими картами.

Пізніше в GSFC був зібраний кластер TheHIVE-Highly-parallel Integrated Virtual Environment, структура якого показана на рис. 3.14. Цей кластер складається з чотирьох підкластерів E, B, G, і DL, об'єднуючи 332 процесори і два виділених хост-комп'ютери. Всі вузли даного кластеру працюють під управлінням Red Hat Linux.

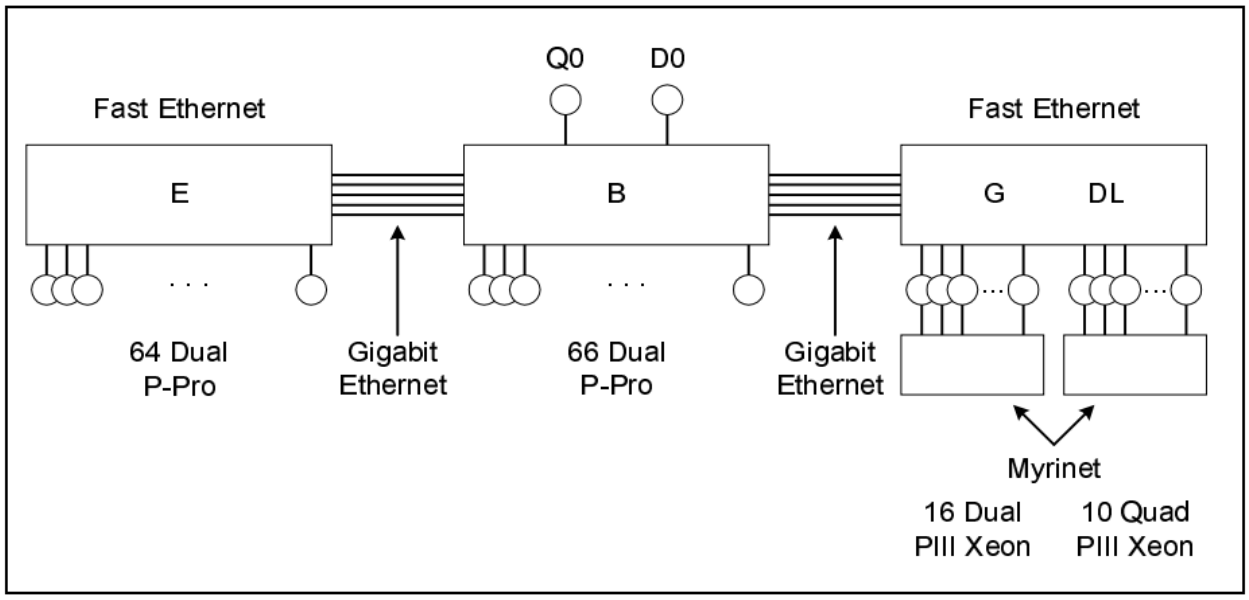


Рисунок 3.14. – Структура кластера TheHIVE

У 1998 році в Лос-Аламоській національній лабораторії на базі процесорів Alpha 21164A з тактовою частотою 533 МГц був створений Linux-кластер Avalon. Спочатку Avalon складався з 68 процесорів, потім їх число було збільшено до 140. У кожному вузлі встановлено по 256 Мбайт оперативної пам'яті, жорсткий диск на 3 Гбайт і мережевий адаптер Fast Ethernet. Загальна вартість проекту Avalon склала 313 тисяч доларів. Показана кластером продуктивність на тесті LINPACK-47,7 Гфлопс, дозволила йому зайняти 114 місце в 12-й редакції списку Top500 поруч з 152-процесорною системою IBM RS/6000 SP. У тому ж 1998 році на самій престижній конференції в області високопродуктивних обчислень Supercomputing'98 творці Avalon представили доповідь "Avalon: An Alpha/Linux Cluster Achieves 10 Gflops for $ 150k", що

отримала першу премію у номінації "Найкраще співвідношення ціна/ продуктивність" ("1998 Gordon Bell Price/Performance Prize").

У квітні 2000 року в рамках проекту АСЗ в Корнельському університеті для проведення біомедичних досліджень був встановлений кластер Velocity +. Він складається з 64 вузлів з чотирьма процесорами Intel Pentium III кожен. Вузли працюють під управлінням Windows 2000 і об'єднані мережею cLAN.

Проект LoBoS (Lots of Boxes on Shelfes) реалізований у Національному Інституті здоров'я США в квітні 1997 року. Він цікавий використанням в якості комунікаційного середовища технології Gigabit Ethernet. Спочатку кластер складався з 47 вузлів з двома процесорами Intel Pentium Pro/200 МГц, 128 Мбайт оперативної пам'яті і диском на 1,2 Гбайт на кожному вузлі. У 1998 році був реалізований наступний етап проекту – LoBoS2, в ході якого вузли були перетворені в настільні комп'ютери із збереженням об'єднання в кластер. Зараз LoBoS2 складається з 100 обчислювальних вузлів, що містять по два процесори Pentium II/450 МГц, 256 Мбайт оперативної і 9 Гбайт дискової пам'яті. Додатково до кластеру підключені 4 керуючих комп'ютера із загальним RAID-масивом об'ємом 1,2 Тбайт.

Цікавою розробкою став у 2000 році кластер KLAT2 (Kentucky Linux Athlon Testbed 2). Система KLAT2 складається з 64 бездискових вузлів з процесором AMD Athlon/700 МГц і оперативною пам'яттю 128 Мбайт на кожному. Програмне забезпечення, компілятори та математичні бібліотеки (SCALAPACK, BLACS і ATLAS) були доопрацьовані для ефективного використання технології 3D-Now! процесорів AMD. Ця робота дозволила істотно збільшити продуктивність системи в цілому. Значний інтерес представляє і використане мережеве рішення, назване "Flat Neighbourhood Network" (FNN). У кожному вузлі встановлено чотири мережевих адаптера Fast Ethernet, а вузли з'єднуються за допомогою дев'яти 32-портових комутаторів. При цьому для будь-яких двох вузлів завжди є пряме з'єднання через один з комутаторів, але немає необхідності в з'єднанні всіх вузлів через єдиний комутатор. Завдяки оптимізації программного забезпечення під архітектуру AMD і топології FNN вдалося домогтися рекордного співвідношення ціна/продуктивність на тесті LIN PACK-650 доларів за 1 Гфлопс.

Ідея розбиття кластера на розділи отримала цікаве втілення у проекті Chiba City, реалізованому в Аргонській Національній лабораторії. Головний розділ містить 256 обчислювальних вузлів, на кожному з яких встановлено по два процесори Pentium III/500 МГц, 512 Мбайт оперативної пам'яті і локальний диск об'ємом 9 Гбайт. Крім обчислювального розділу в систему входять: розділ візуалізації (32 комп'ютери IBM Intellistation з графічними картами Matrox Millenium G400, 512 Мбайт оперативної пам'яті і дисками 300 Гбайт), розділ зберігання даних (8 серверів IBM NetFinity 7000 з процесорами Xeon/500 МГц і дисками по 300 Гбайт) і керуючий розділ (12 комп'ютерів IBM Netfmity 500). Всі вони об'єднані мережею Myrinet, яка використовується для підтримки паралельних додатків. Для керуючих і службових цілей використовуються мережі Gigabit Ethernet і Fast Ethernet. Всі розділи діляться на "міста" (towns) по 32 комп'ютера. Кожен з них має свого "мера", який локально обслуговує своє

"місто", знижуючи навантаження на службову мережу і забезпечуючи швидкий доступ до локальних ресурсів.

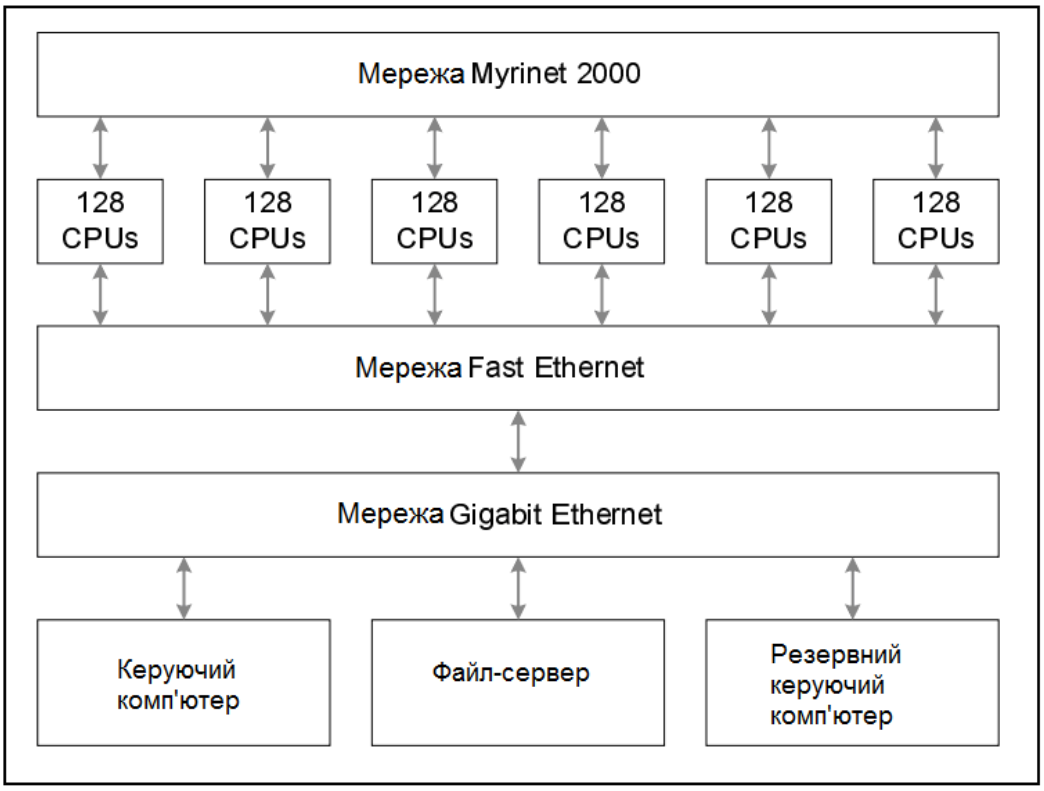


Рисунок 3.15. – Структура суперкомп’ютера МВС-1000М

Кластером є і суперкомп’ютер МВС-1000М, встановлений у Міжвідомчому суперкомп’ютерному центрі в Москві (http://www.jscc.ru). Комп’ютер складається з шести базових блоків, що містять по 64 дво-процесорних модулі (рис. 3.15). Кожен модуль має два процесори Alpha 21264/667 МГц (кеш-пам’ять другого рівня 4 Мбайт), 2 Гбайт оперативної пам’яті, що розділяється процесорами модуля, жорсткий диск. Загальне число процесорів у системі дорівнює 768, а пікова продуктивність МВС-1000М перевищує 1 Тфлопс.

Всі модулі МВС-1000М пов’язані двома незалежними мережами. Мережа Myrinet 2000 використовується програмами користувачів для обміну даними в процесі обчислень. При використанні MPI пропускна здатність каналів мережі досягає значень 110-170 Мбайт/с. Мережа Fast Ethernet використовується операційною системою для виконання сервісних функцій.

3.6. Комунікаційні технології побудови кластерів

Зрозуміло, що різних варіантів побудови кластерів дуже багато. Одна з істотних відмінностей лежить в використовуваній мережевій технології, вибір якої визначається, перш за все, класом вирішуваних завдань.

Спочатку Beowulf-кластери будувалися на базі звичайної 10-мегабітної мережі Ethernet. Сьогодні часто використовується мережа Fast Ethernet, як правило, на базі комутаторів. Основна перевага такого рішення – це низька

вартість. Разом з тим, великі накладні витрати на передачу повідомлень в рамках Fast Ethernet призводять до серйозних обмежень на спектр задач, ефективно вирішуються на таких кластерах. Якщо від кластера потрібна велика універсальність, то потрібно переходити на інші, більш продуктивні комунікаційні технології. Виходячи з міркувань вартості, продуктивності і масштабованості, розробники кластерних систем роблять вибір між Fast Ethernet, Gigabit Ethernet, SCI, Myrinet, cLAN, ServerNet і рядом інших мережевих технологій (основні параметри комунікаційних технологій можна знайти, наприклад, на http:// www.Parallel.ru).

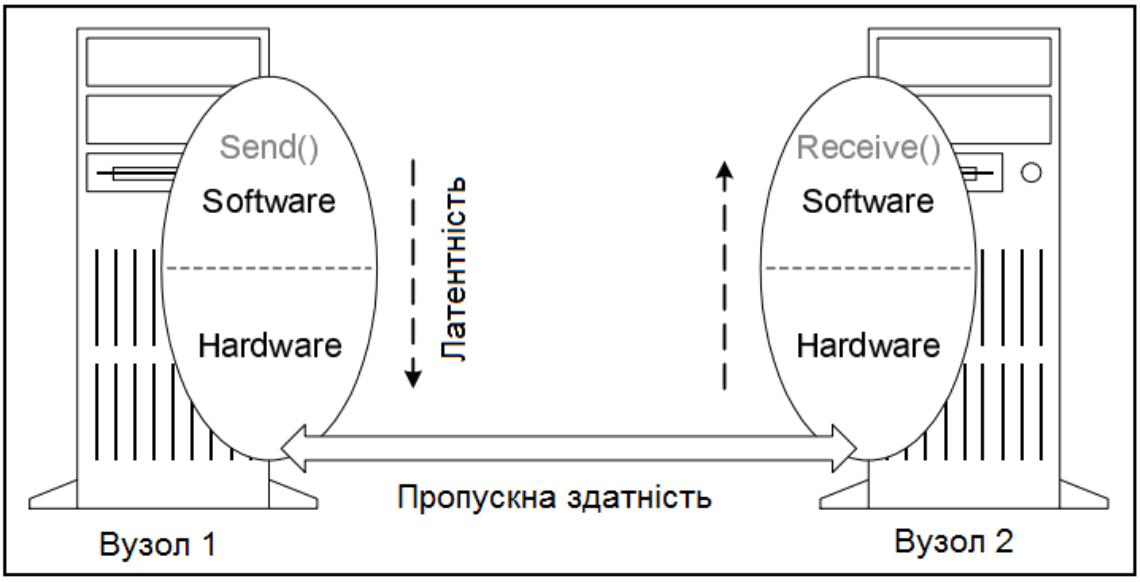


Рисунок 3.16. – Латентність і пропускна здатність комунікаційного середовища

Якими ж числовими характеристиками виражається продуктивність комунікаційних мереж в кластерних системах? Необхідних користувачу характеристик дві: латентність і пропускна здатність мережі. Латентність – це час початкової затримки при посилці повідомлень. Пропускна здатність мережі визначається швидкістю передачі інформації по каналах зв'язку (Рис. 3.16). Якщо в програмі багато маленьких повідомлень, то сильно позначиться латентність. Якщо повідомлення передаються більшими порціями, то важлива висока пропускна здатність каналів зв'язку. Через латентність максимальна швидкість передачі по мережі не може бути досягнута на повідомленнях з невеликою довжиною.

На практиці користувачам не стільки важливі пікові характеристики, що заявляються виробником, скільки реальні показники, що досягаються на рівні додатків. Після виклику користувачем функції відправки повідомлення Send () повідомлення послідовно проходить через цілий набір шарів, визначених особливостями організації програмного забезпечення та апаратури. Цим, зокрема, визначаються і безліч варіацій на тему латентності реальних систем. Встановили MPI на комп'ютері погано, латентність буде велика, купили дешеву мережеву карту від невідомого виробника, чекайте подальших сюрпризів.

На закінчення параграфа давайте спробуємо і для даного класу комп’ютерів виділити чинники, що знижують продуктивність обчислювальних систем з розподіленою пам’яттю на реальних програмах.

Почнемо з уже згадуваного раніше закону Амдала. Для комп’ютерів данного класу він грає дуже велику роль. Справді, якщо припустити, що в програмі є лише 2% послідовних операцій, то розраховувати на більш ніж 50-кратне прискорення роботи програми не доводиться. Тепер спробуйте критично поглянути на свою програму. Швидше за все, в ній є ініціалізація, операції введення/виведення, якісь суто послідовні ділянки. Оцініть їх частку на тлі всієї програми і на мить припустіть, що ви отримали доступ до обчислювальної системи з 1000 процесорів. Після обчислення верхньої межі для прискорення програми на такій системі, думаємо, стане ясно, що недооцінювати вплив закону Амдала ніяк не можна.

Оскільки комп'ютери даного класу мають розподілену пам'ять, то взаємодія процесорів між собою здійснюється за допомогою передачі повідомлень. Звідси два інших уповільнюючих чинника – латентність і швидкість передачі даних по каналах комунікаційного середовища. В залежності від комунікаційної структури програми ступінь впливу цих факторів може сильно змінюватися.

Якщо апаратура або програмне забезпечення не підтримують можливості асинхронної відправки повідомлень на тлі обчислень, то виникнуть неминучі накладні витрати, пов'язані з очікуванням повного завершення взаємодії паралельних процесів.

Для досягнення ефективної паралельної обробки необхідно домогтися максимально рівномірного завантаження всіх процесорів. Якщо рівномірності немає, то частина процесорів неминуче буде простоювати, очікуючи інших, хоча в цей час вони цілком могли б виконувати корисну роботу. Дана проблема вирішується простіше, якщо обчислювальна система однорідна. Дуже великі труднощі виникають при переході на неоднорідні системи, в яких є значна різниця або між обчислювальними вузлами, або між каналами зв'язку.

Істотний фактор – це реальна продуктивність одного процесора обчислювальної системи. Різні моделі мікропроцесорів можуть підтримувати кілька рівнів кеш-пам'яті, мати спеціалізовані функціональні пристрої і т. д. Візьмемо хоча б ієрархію пам'яті комп'ютера Cray T3E: регістри процесора, кеш-пам'ять 1-го рівня, кеш-пам'ять 2-го рівня, локальна пам'ять процесора, дистанційна пам'ять іншого процесора. Ефективне використання такої структури вимагає особливої уваги при виборі підходу до вирішення завдання.

Додатково кожен мікропроцесор може мати елементи векторно-конвеєрної архітектури.

На жаль, як і колись, на роботі кожної конкретної програми в тій чи іншій мірі позначаються всі ці фактори. Однак на відміну від комп'ютерів інших класів, сумарний вплив викладених тут факторів може знизити реальну продуктивність не в десятки, а в сотні і навіть тисячі разів у порівнянні з піковою. Потенціал комп'ютерів цього класу величезний, домогтися на них можна дуже багато чого. Однак можуть знадобитися значні зусилля. Всі етапи у

вирішенні кожної задачі, починаючи від вибору методу до запису програми, потрібно продумувати дуже акуратно.

Крайня точка – Інтернет. Його теж можна розглядати як комп'ютер з розподіленою пам'яттю. Причому, як найпотужніший у світі комп'ютер. Спробуйте знайти спосіб вирішення ваших завдань на такому комп'ютері.

3.7 Методи оцінювання продуктивності суперкомп'ютерів

Більшість сьогоднішніх оцінкових характеристик продуктивності суперкомп'ютерів пов'язані з оцінюванням швидкості виконання ними обчислень. Насамперед до них належить пікова продуктивність, вимірювана в мільйонах операцій з рухомою комою, які комп'ютер теоретично може виконати за 1 секунду *(FLOPS-floating-point operations per second).* Пікова продуктивність – величина, практично недосяжна. Це пов'язано зокрема з проблемами заповнення функціональних конвеєрних пристроїв, що є типовим не тільки для векторних суперкомп'ютерів, але і для комп'ютерів на основі мікропроцесорів *RISC*-архітектури. Особливо важливо це для суперконвеєрної архітектури мікропроцесорів, наприклад, *DEC Alpha,* для якої характерне застосування відносно довгих конвеєрів. Зрозуміло, що чим довшим є конвеєр, тим більше треба "ініціалізаційного" часу для того, щоб його заповнити. Такі конвеєри ефективні під час роботи з довгими векторами. Тому для оцінювання векторних суперкомп'ютерів було введено таке поняття, як довжина напівпродуктивності – довжина вектора, за якої досягають половини пікової продуктивності.

Варто зазначити, що продуктивність виконання операцій з рухомою комою можна вказати як для виконання обчислень над даними у форматі з одинарною точністю, так і для виконання обчислень над даними у форматі з подвійною точністю. Термін "подвійна точність" вживають тоді, коли обчислення виконують над словами даних, кожне з яких займає дві сусідні комірки у пам'яті комп'ютера. Формат даних, поданих із подвійною точністю, інколи називають подвійним *(double),* числа в цьому форматі можуть бути визначені як цілі, з фіксованою або рухомою комою. Стандарт *IEEE 754* визначає формат подання даних з подвійною точністю з рухомою комою. Дані в цьому форматі є 64-розрядними.

Реальніші оцінки продуктивності грунтуються на часах виконання різних тестів. Звичайно ж, найкращими тестами є реальні завдання користувача. Однак такі оцінювання, по-перше, є дуже специфічними, а по-друге, часто взагалі недоступні або відсутні. Тому зазвичай застосовують універсальні тести, однак традиційні методики оцінювання продуктивності мікропроцесорів – *SPEC (Standard Performance Evaluation Corporation*) – для суперкомп'ютерів використовують рідко, хоча розроблено спеціальні стандарти *SPEChpc96* і *SPEChpc2002* для високопродуктивних обчислень. Сьогодні є

оцінювання *SPEC* тільки для суперкомп'ютерів, що використовують мікропроцесори *RISK*-архітектури. Сама організація *SPEC* припинила продаж програмних пакетів *SPEChpc2002* у лютому 2007 р., відтоді ж не публікує результатів виконання цих тестів і не надає технічного супроводу.

Оскільки більшу частину часу виконання програм зазвичай займають цикли, іноді саме їх застосовують як тести – наприклад, відомі ліверморські цикли. Най популярнішим тестом продуктивності сьогодні визнано Linpack, який являє собою розв'язання системи / лінійних рівнянь за методом Гаусса. Оскільки відомо, скільки операцій з числами потрібно виконати для розв'язання системи та знаючи час розрахунку, можна обчислити кількість операцій, виконуваних за секунду. Є кілька модифікацій цих тестів. Зазвичай фірми-виробники комп'ютерів наводять результати при N=100. Вільно поширюється стандартна програма мовою Фортран, яку треба виконати на суперкомп'ютері, щоб отримати результат тестування. Цю програму не можна змінити, за винятком заміни викликів підпрограм, що дають доступ до процесорного часу виконання. Інший стандартний тест належить випадку N= 1000, що передбачає використання довгих векторів. Ці тести можна виконувати на комп'ютерах за різної кількості процесорів, даючи також оцінки якості розпаралелювання. Для багатопроцесорних систем цікавішим є тест *Linpack-parallel,* за яким продуктивність вимірюється за великих значень / і кількості процесорів. Для високопаралельних суперкомп'ютерів сьогодні все більше використовують тести *NAS parallel benchmark,* розроблені *NASA Advanced Supercomputing (NAS) Division* (раніше *NASA Numerical Aerodynamic Simulation Program).*

3.8. Побудова та характеристики сучасних суперкомп'ютерів

У сучасних суперкомп'ютерах, наприклад, в суперкомп'ютерах фірм *Cray, NEC, Fujitsu, Hitachi,* найчастіше застосовується модифікована структура ОКМД. Нижче наведено ряд прикладів, що показують основні характеристики сучасних суперкомп'ютерів цього класу.

3.8.1. Суперкомп'ютер CRAY T932

CRAY T932 (рис. 3.17) – векторно-конвеєрний суперкомп'ютер фірми *CRAY Research Inc.* (сьогодні це підрозділ *Silicon Graphics Inc.),* уперше випущений у 1996 році. Максимальна продуктивність одного процесора дорівнює майже 2 млрд операцій за секунду, основна пам'ять нарощується до 8 Гб, дисковий простір – до 56000 Гб (тобто 256 Тб).

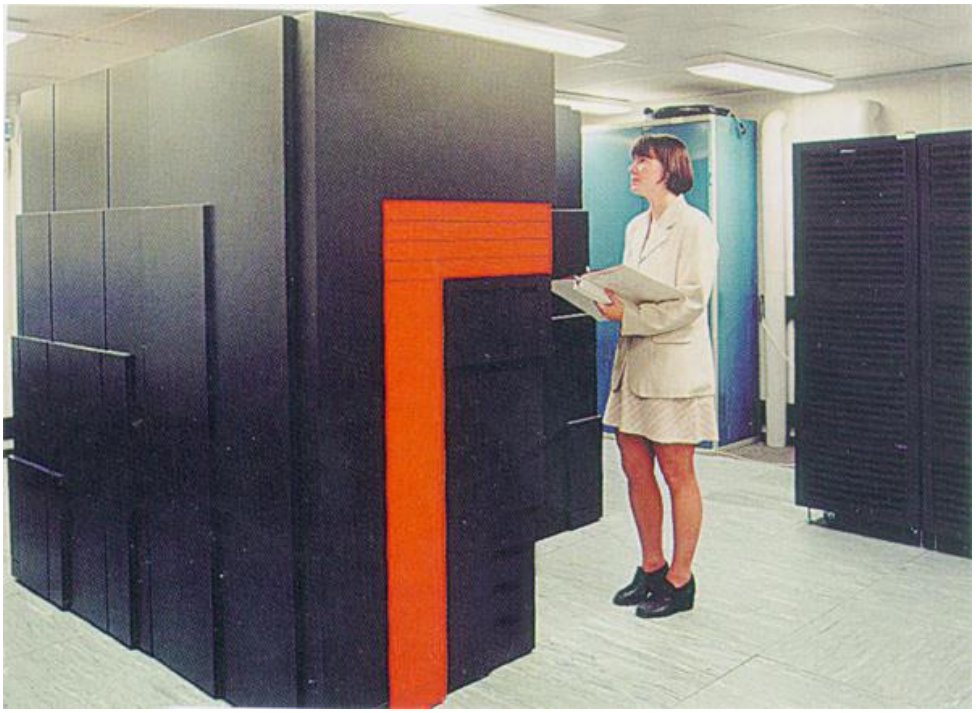


Рисунок 3.17. – Зовнішній вигляд суперкомп'ютера 7932 фірми ***CRAY***

Комп'ютер у максимальній конфігурації вміщує 32 процесори, що працюють і загальною пам'яттю, тому максимальна продуктивність всієї комп'ютерної системи становить понад 60 млрд операцій за секунду.

3.8.2. Суперкомп'ютер IBM SP2

IBM SP2 (рис. 3.18) – матричний паралельний суперкомп'ютер фірми *IBM,* виготовлений у 2006 р.

Суперкомп'ютер *SP2* побудований на основі стандартних процесорів *PowerPC 604e* або *POWER2 SC,* сполучених між собою через високошвидкісний комутатор, причому кожен з них має свою локальну основну пам'ять і дискову підсистему. Характеристики цих процесорів відомі й особливого подиву не викликають, проте у межах однієї системи *SP2* їх може бути об'єднано дуже багато. Зокрема, максимальна система, встановлена в *Pacific Northwest National Laboratory* (Річленд, США), вміщує 512 процесорів. За кількістю процесорів можна розрахувати сумарну потужність всієї обчислювальної системи.

Рисунок 3.18. – Зовнішній вигляд суперкомп'ютера SP2 фірми IBM

3.8.3. Суперкомп'ютер HP Exemplar

Суперкомп'ютер *HP Exemplar* – комп'ютер із кластерною архітектурою, який виготовила компанія *Hewlett-Packard Inc* у 1998 р. Модель *V2250* (клас *V)* побудована на основі мікропроцесора *PA-8200,* що працює на тактовій частоті 240 МГц. У межах одного вузла із загальною основною пам'яттю до 16 Гб можна об'єднати до 16 процесорів. Своєю чергою, вузли у межах однієї комп'ютерної системи з'єднуються між собою високошвидкісними каналами передавання даних.

3.8.4. Суперкомп'ютер Intel ASCI RED

Суперкомп'ютер *ASCI RED* (рис. 3.19) виготовила компанія *Intel* у результаті виконання програми *Accelerated Strategic Computing Initiative* у 1997 р. на замовлення Міністерства енергетики США. Він об'єднує 9152 процесорів *Pentium Pro,* має 600 Гб сумарної основної пам'яті та загальну продуктивність 1800 мільярдів операцій за секунду (1,8 TFLOPS).

Рисунок 3.19. – Зовнішній вигляд суперкомп'ютера ASCI RED фірми Intel

3.8.5. Суперкомп'ютер IBM Blue Gene/L

Суперкомп'ютер фірми *IBM Blue Gene/L* (рис. 3.20), виготовлений у 2006 р., містив 131072 процесорних вузлів. Його теоретична пікова продуктивність становила 360 TFLOPS, а реальна продуктивність, отримана на тесті *Linpack*, дорівнювала 280,6 TFLOPS. Після оновлення в 2007 році реальна продуктивність збільшилася до 478 TFLOPS за пікової продуктивності у 596 TFLOPS. Кожен вузол містить процесор *Power PC 440* з 512 MB локальної пам'яті.

Рисунок 3.20. – Зовнішній вигляд суперкомп'ютера *Blue Gene/L* фірми *IBM*

3.8.6. Суперкомп'ютер RIKEN MDGRAPE-3

Також у 2006 році було введено в експлуатацію суперкомп'ютер *MDGRAPE*-3, який досяг продуктивності 1,015 PFLOPS (PFLOPS – петафлопс; 1 PFLOPS = 1000 GFLOPS). Суперкомп'ютер *MDGrape-3* (рис. 3.21) розробили

в японському дослідному інституті *RIKEN* в м. Йокогама за технічної підтримки компанії *SGI Japan* та японського підрозділу компанії *Intel,* для розв'язання задач молекулярної динаміки, а саме моделювання згортання білків.

Рисунок 3.21. – Зовнішній вигляд суперкомп'ютера MDGRAPE-3 інституту RIKEN

Як основні складові комп'ютера *MDGrape-3* було використано процесор *MDGrape-3* фірми *Hitachi,* які забезпечують продуктивність 230 GFLOPS кожен. Один обчислювальний модуль містив 24 таких процесори, а вся система складалася з 201 модуля.

Роботу головного обчислювального ядра забезпечували дві групи серверів і основі процесорів *Intel,* кластер з 64 паралельних серверів з двоядерним процесором (разом 256 процесорних ядер) *Intel Xeon (Dempsey),* і ще один кластер з 37 серверів (74 одноядерних процесори *Intel Xeon* 3,2 Ггц). Всі ці комп'ютери у єдину систему масового паралелізму об'єднала компанія *SGI Japan.*

На момент введення в експлуатацію *MDGrape-3* за продуктивністю майя втричі перевершував описаний вище американський суперкомп'ютер IB *BlueGene/L,* що тоді був визнаний найпродуктивнішим. Однак внаслідок того, її *MDGrape-3* не був комп'ютером загального призначення, вимірювання його продуктивності за допомогою тесту *Linpack* (розв'язання великих систем лінійні рівнянь) ускладнювалось, а відповідно, цей комп'ютер не було внесено до списі "найпотужніших" за рейтингом TOP-500.

3.8.7. Суперкомп'ютер IBM Roadrunner

Суперкомп'ютер *IBM Roadrtinner* створили фахівці компанії *IBM* і JTос-Аламоської національної лабораторії США у 2008 р. Система *Roadrunner* (рис. 3.22), у якій використовуються процесори *Cell* і *AMD Opteron,* продемонструвала сталу продуктивність понад один петафлоп, а саме 1,026 PFLOPS у липні 2008 року і 1,105 PFLOPS у листопаді 2008 року.

Рисунок 3.22. – Зовнішній вигляд суперкомп'ютера *Roadrunner* фірми *IBM*

Суперкомп'ютер *Roadrunner* призначався для виконання досліджень у галузі ядерного озброєння, зокрема для моделювання ядерних вибухів, однак протягом перших шести місяців експлуатації він виконував завдання інших галузей: від розроблення біопалива і проектування ефективних двигунів до фармацевтичних досліджень і фінансового моделювання. Надалі 75 % часу він був задіяний у військових програмах.

Щоб проілюструвати колосальну продуктивність *Roadrunner,* що дорівнює продуктивності 100 тисяч тогочасних ноутбуків, компанія *IBM* наводила таке порівняння: обсяг обчислень, який могли б виконати всі люди на Землі (шість мільярдів чоловік) за допомогою калькуляторів, роблячи одне обчислення кожну секунду без перерви протягом 46 років, дорівнює кількості обчислень, виконуваних новим суперкомп'ютером за один день.

До конфігурації *Roadrunner* входить 6948 двоядерних процесорів *AMD Opteron* і 12960 процесорів *Cell,* 80 ТБ оперативної пам'яті. Зазначене обладнання розміщено в 288 шафах і з'єднано сотнею кілометрів оптичного волокна. Споживана потужність *Roadrunner* становить 3,9 МВт, що тоді робило систему найбільш енергетично ефективною в світі – вона виконувала 376 млн обчислень у розрахунку на один Вт електроенергії. Вартість *IBMRoadrunner* перевищувала 100 млн доларів.

Згодом компанія *IBM* оголосила про свій намір підкорити наступну висоту – один екзафлоп (exa-FLOP). Анонсувалось, що відповідну систему буде створено в Дубліні у межах співпраці з агентством з промислового розвитку Ірландії. Екзафлоп – величина, що дорівнює мільйону трильйонів обчислень з рухомою комою за секунду, що в 1000 разів більше, ніж досягнуте значення одного петафлопа.

3.8.8. Суперкомп'ютери Tianhe-1 та Tianhe-1A Національного університету оборонних технологій Китаю

У 2009 році фахівцям Національного університету оборонних технологій Китаю вдалося повторити досягнення американських колег. Вони створили

суперкомп'ютер під назвою *Tianhe-1 (Tianhe* – англ. "*Milky way",* укр. "Чумацький шлях") (рис. 3.23), продуктивність якого перевищила 1 PFLOPS. Точніше кажучи, пікова продуктивність системи досягає 1,206 PFLOPS, продуктивність у сталому режимі – 563,1 TFLOPS (за результатами тесту *Unpack).* Тобто, Китай став другою країною, де було створено настільки високопродуктивний комп'ютер.

Рисунок 3.23. – Зовнішній вигляд суперкомп'ютера *Tianhe-1* Національного університету оборонних технологій Китаю

До конфігурації китайського "Чумацького шляху" входить 6144 чотириядерні процесори *Intel* (3072 *XeonE5540* і стільки ж *Xeon E5450)* та 5120 графічних процесорів *AMD* (2560 двопроцесорних карт *Radeon HD 4870 X2* з 1 ГБ пам'яті *GDDR5* кожна). Систему вагою 155 тонн змонтовано в 103 шафах, що займають площу близько 1000 кв.м. Витрати на створення суперкомп'ютера оцінюються приблизно в 88 млн доларів США.

На виставці *HPC 2010 China,* яка проходила з 27 по 30 жовтня 2010 року в Пекіні, було представлено оновлений суперкомп'ютер під назвою *Tianhe-1A.*

(Тяньхе-1А). Згідно з тестами *Unpack,* цей комп'ютер показав рекордну продуктивність в операціях з рухомою комою в 2,507 PFLOPS. Суперкомп'ютер *Tianhe-1A* знаходиться в Національному суперкомп'ютерному центрі в Тяньцзіні.

В основу *Tianhe-IA* покладно сучасну парадигму гетерогенних обчислень – суперкомп'ютер містить 7168 паралельно працюючих графічних процесорів *NVIDIA Tesla M2050* і 14366 центральних процесорів *Intel Xeon. Якби Tianhe-1A* використовував тільки центральні процесори без графічних, то йому б знадобилося для досягнення такої самої продуктивності понад 50 тис. процесорів, при цьому площа займаного комп'ютером простору збільшилася б удвічі.

Графічні процесори *NVIDIA Tesla,* що використовують архітектуру паралельних обчислень *CUDA,* застосовуються для створення високопродуктивних обчислювальних середовищ і забезпечують потужний приріст продуктивності для широкого спектра завдань, зокрема у фармацевтиці,

для моделювання ураганів і цунамі, проектування автомобілів і навіть вивчення формування галактик.

Більше того, без графічних процесорів 2,5-петафлопсний комп'ютер характеризувався б величезною споживаної потужністю – не менше 12 МВт. Завдяки використанню гетерогенної архітектури розробникам вдалося знизити енергоспоживання втричі – до 4,04 МВт.

3.8.9. Суперкомп'ютер Cray Jaguar XT5

Суперкомп'ютер *Jaguar XT5* (рис. І.І3), виготовлений компанією *Cray*, було визнано найпродуктивнішим у листопаді 2009 року. За продуктивністю 1,75 PFLOPS, він випередив *IBM Roadrunner.*

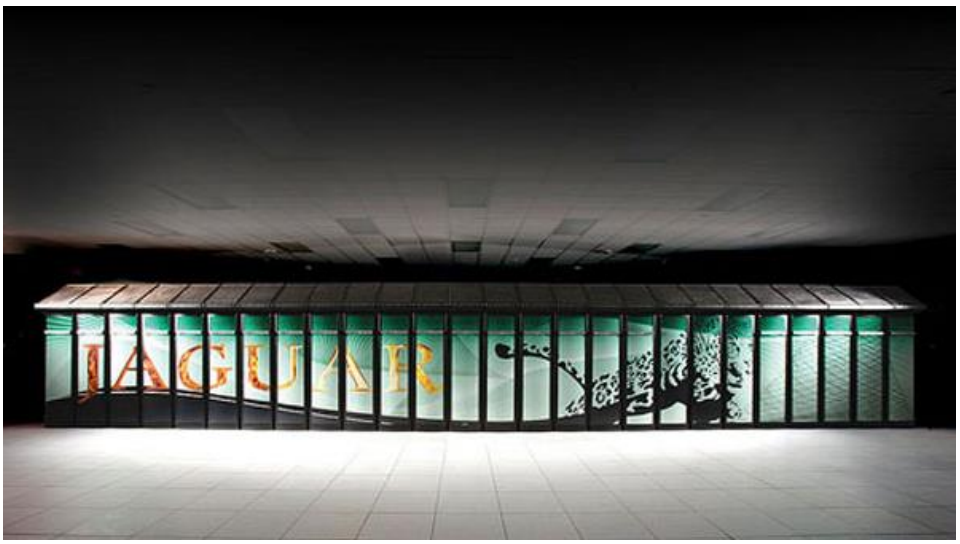


Рисунок 3.24. – Зовнішній вигляд суперкомп'ютера *Jaguar XT5* фірми *Cray*

Суперкомп'ютер *Jaguar* належить до класу систем масового паралелізму. Він складається з множини автономних вузлів (англ. *nodes).* Вузли є двох типів: *XT5* і *XT4* для моделей суперкомп'ютера *Cray Jaguar XT5* і *Jaguar XT4,* відповідно.

Суперкомп'ютер *XT5* містить 18 688 обчислювальних вузлів, а також допоміжні вузли для входу користувачів та обслуговування. Кожен обчислювальний вузол містить 2 чотириядерні процесори *AMD Opteron 2356 (Barcelona*) з внутрішньою частотою 2,3 ГГц, 16 ГБ пам'яті *DDR2-800* і маршрутизатор *SeaStar 2+*. Загалом суперкомп'ютер *XT5* містить 149504 обчислювальні ядра, понад 300 ТБ пам'яті, понад 6 ПБ дискового простору і має пікову продуктивність 1,38 PFLOPS. У період з липня до листопада 2009 року відбулося оновлення вузлів суперкомп'ютера *Jaguar XT5:* 4-ядерні процесори *Opteron* було замінено на 6-ядерні, завдяки чому вдалося підняти продуктивність суперкомп'ютера до згаданого показника в 1,75 PFLOPS.

Суперкомп'ютер *XT4* містить 7832 обчислювальні вузли, а також допоміжні вузли для входу користувачів та обслуговування. Вузол містить чотириядерний процесор *AMD Opteron 1354 (Budapest*) з внутрішньою частотою 2,1 ГГц, 8 Гб пам'яті *DDR2-800* (в деяких комірках – *DDR2-667)* і

маршрутизатор *SeaStar2*. Загалом суперкомп'ютер *XT4* містить 31328 обчислювальних ядер, понад 62 ТБ пам'яті, понад 600 ТБ дискового простору і має пікову продуктивність 263 TFLOPS.

3.8.10. Суперкомп'ютер K Computer компанії Fujitsu та Інституту фізико-хімічних досліджень RIKEN

На 26-й Міжнародній конференції із Суперкомп'ютерів (*International Super- computing Conference),* що проходила у Гамбурзі (Німеччина) в червні 2011 року, було продемонстровано суперкомп'ютер під назвою *K Computer* виробництва японської компанії *Fujitsu,* який запущено в 2011 році в Інституті фізико-хімічних досліджень *RIKEN Advanced Institute for Computational Science* в місті Кобе (Японія). Назва суперкомп'ютера походить від японського слова "*Kei*" (Кей), що означає Юпета (10×10^{15}). Цей комп'ютер у червні 2011 року очолив рейтинг TOP500 найпотужніших суперкомп'ютерів світу [2]. За результатами тесту *Unpack,* показник продуктивності *K Computer* становить 8,16 петафлопс. За продуктивністю *K Computer* випереджає попереднього лідера – китайський суперкомп'ютер *Tianhe-IA* – більш ніж утричі.

На рис. 3.25 показано зовнішній вигляд суперкомп'ютера *K Computer* станом на червень 2011 року (матеріали сайту *http://www.nsc.riken.jp).*

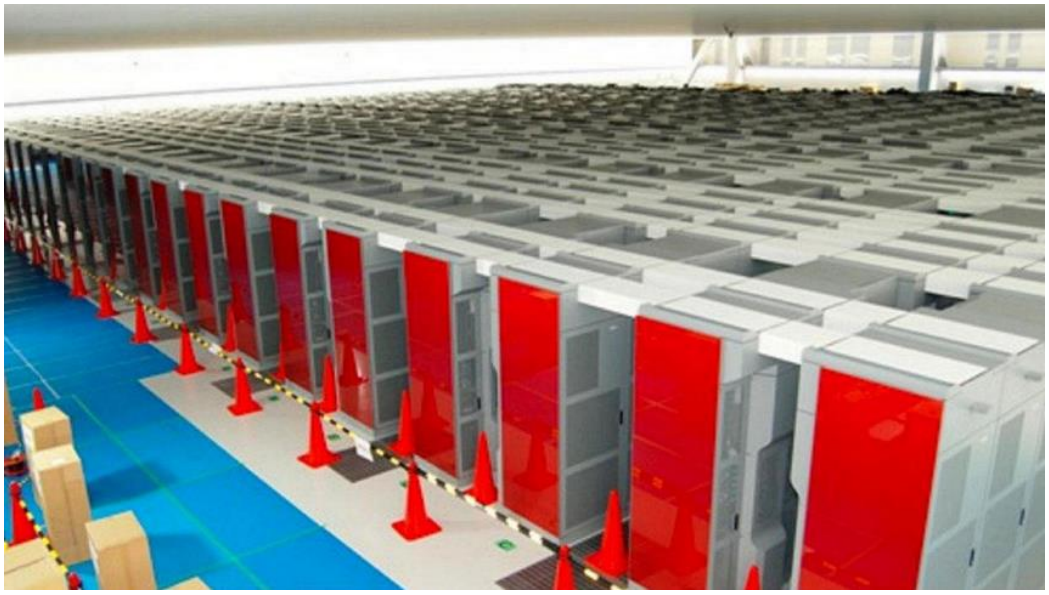


Рисунок 3.25. – Зовнішній вигляд суперкомп'ютера *K Computer* станом на червень 2011 року

На той час система складалась із 672 комп'ютерних стійок і налічувала 68544 восьмиядерні процесори *SPARC64 Vlllfx* (що разом становить 548352 обчислювальні ядра), які виготовила за 45-нанометровим технологічним процесом компанія *Fujitsu.* Продуктивність одного такого процесора дорівнює 128 Гфлопс. Важливо зазначити, що обчислювальна ефективність (відношення середньої продуктивності до пікової продуктивності) суперкомп'ютера становила 93 % [3].

Суперкомп'ютер використовує водяне охолодження, що дає змогу знизити споживання електроенергії. Зазначимо, що загальна споживана потужність суперкомп'ютера на червень 2011 року становила 9.89 МВт, тоді як середня споживана потужність десяти найпродуктивніших суперкомп'ютерів за версією TOP500 дорівнює 4.3 МВт. Разом з тим, відношення продуктивності су пер коми' юте р а до споживаної потужності робить його одним з найбільш енергоефективних суперкомп'ютерів у світі (шосте місце у рейтингу *http://www.green500.org).*

У листопаді 2011 року суперкомп'ютер *К Computer* підтвердив статус найпотужнішого, продемонструвавши продуктивність 10,51 Петафлопс, причому його пікова продуктивність становить 11,28 Петафлопс. Роботи над створенням суперкомп'ютера було закінчено. В результаті система складається з понад 800 стійок і містить 705024 процесорних ядер (кількість процесорів становить 88128). Вартість проекту *К Computer* – приблизно 1.4 млрд доларів США. Система призначена для виконання наукових розрахунків у таких сферах, як фармацевтика, створення нових матеріалів, кліматичні дослідження, метеорологія, попередження про стихійні лиха, автомобілебудування й астрономія.

Розглянемо детальніше особливості будови суперкомп'ютера *К Computer*. Як було зазначено вище, система складається з 672 комп'ютерних стійок. Будову стійки показано на рис. 3.26.

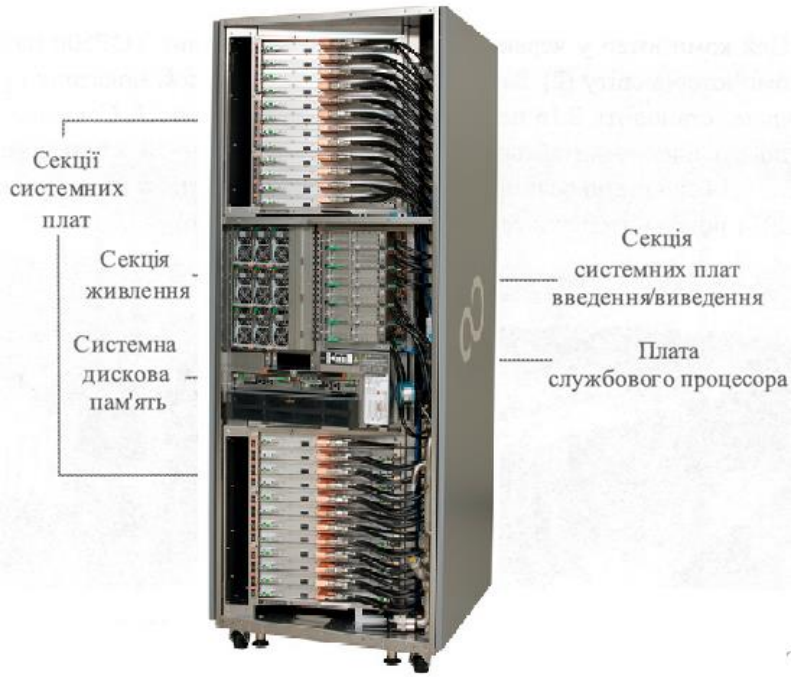


Рисунок 3.26. – Будова стійки суперкомп'ютера *К Computer*

Стійка містить такі складові [4]:

- Секція системних плат;
- Секція живлення;
- Системна дискова пам'ять;
- Секція системних плат введення/виведення;

– Плата службового процесора;
– Система охолодження.

Системна плата (рис. 3.27*)* комплектується чотирма восьмиядерними процесорами *SPARC64™ Vlllfx*. За твердженням виробника [4], плата характеризується найвищим відношенням продуктивності до споживаної потужності, що становить 2,2 Гфлопс/Вт. Системну плату обладнано системою водяного охолодження, яка забирає тепло від процесорів та мережного контролера (*ICC – interconnect controller),* що зменшує енергоспоживання та продовжує термін служби цих компонент.

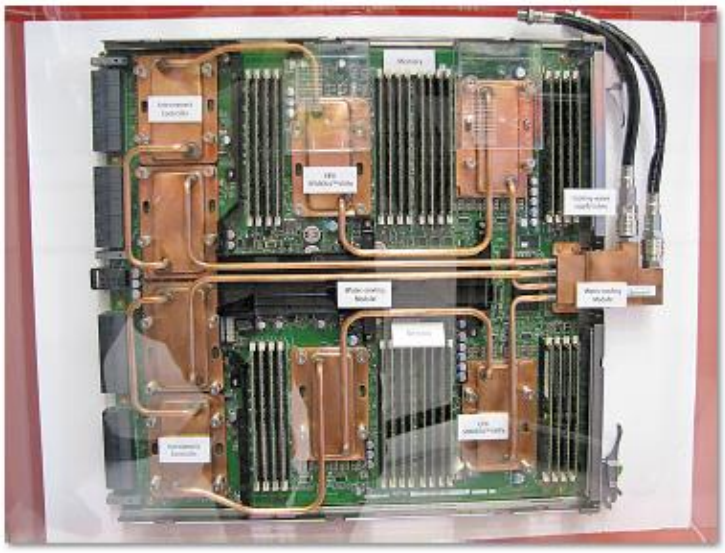
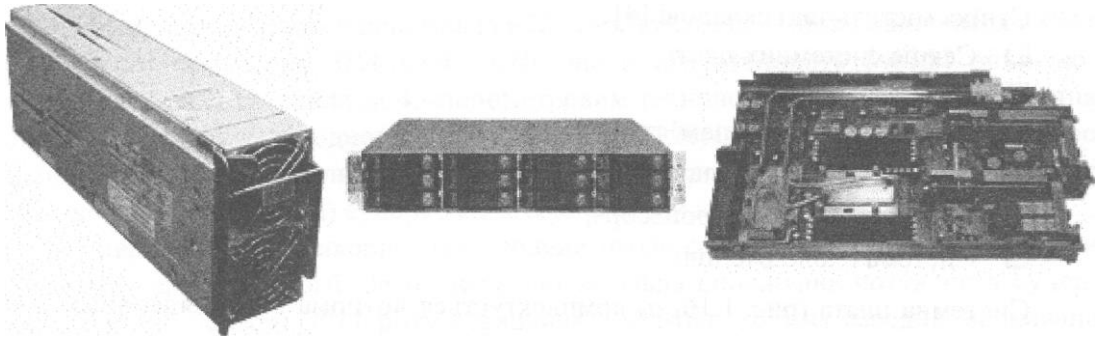


Рисунок 3.27. – Системна плата суперкомп'ютера *K Computer*

Плата службового процесора керує функціонуванням комп'ютерної стійки, виконує ініціалізацію системних плат та виявляє помилки. Для підвищення надійності кожна комп'ютерна стійка містить дві плати службового процесора, одна з яких є резервною.

Стійка містить дев'ять пристроїв живлення (рис. 3.28, а), деякі з яких є резервними. Така кількість забезпечує дуже високу надійність системи живлення і гарантує роботоздатність стійки у разі виходу з ладу декількох пристроїв живлення.



а *б* *в*

Рисунок 3.28. – Пристрій живлення *(а),* системний диск *(б)* та системна плата введення/виведення *(в)* стійки суперкомп'ютера *K Computer*

Системна дискова пам'ять (рис. 3.28, б) використовується для зберігання операційної системи. До стійки також можна під'єднати й додаткову зовнішню пам'ять, наприклад, для завантаження даних та отримання результатів обчислень.

Обмін даними, над якими виконуються обчислення, та результатами обчислень з іншими комп'ютерними стійками та з зовнішніми пристроями зберігання даних забезпечують шість системних плат введення/виведення (рис. 3.28, в).

Вода для охолодження процесорів та мережного контролера системної плати постачається спеціальними трубами системи охолодження (рис. 3.29), які обладнані сенсорами для моніторингу тиску води, температури та ін.

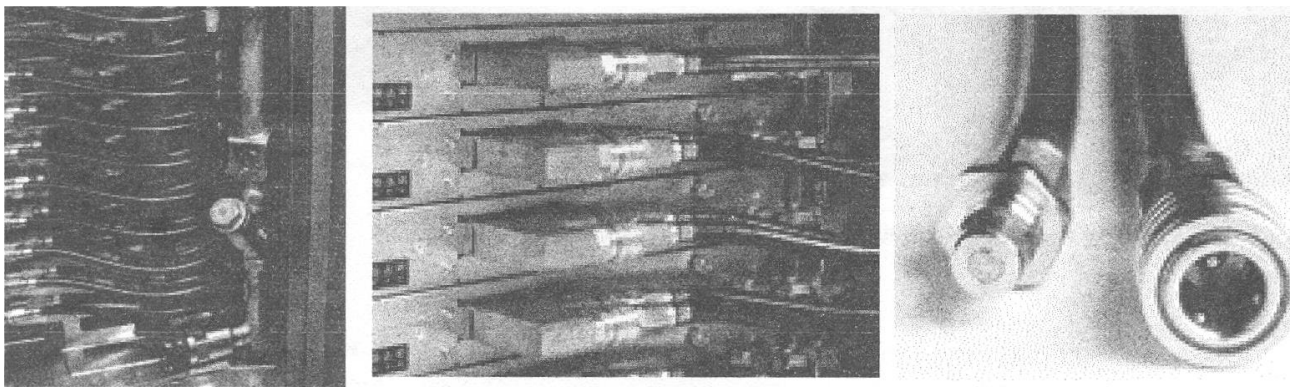


Рисунок 3.29. – Елементи системи охолодження стійки суперкомп'ютера *K Computer*

Разом з водяним застосовується й повітряне охолодження, причому для покращення повітрообміну системні плати в секціях встановлюються не горизонтально, а під певним кутом.

На завершення огляду характеристик сучасних суперкомп'ютерів зазначимо, що галузь високопродуктивних обчислювальних систем є пріоритетною в багатьох розвинених країнах світу, роботи зі створення таких систем ведуться постійно, в результаті чого з'являються нові потужні суперкомп'ютери. Вже під час підготовки книги до друку, в червні 2012 року, було оновлено рейтинг найпотужніших суперкомп'ютерів *TOP500,* у якому першу позицію посів американський суперкомп'ютер *Sequoia – BlueGene/Q* компанії *IBM.* Його реальна продуктивність на тесті *Unpack* становить 16,324 Петафлопс, а пікова – понад 20 Петафлопс. Споживана потужність суперкомп'ютера становить 7,89 МВт, що говорить про його високу енергоефективність. Суперкомп'ютер *Sequoia* побудовано на основі 17-ядерних процесорів *Power A2* (одне ядро слугує для виконання завдань операційної системи, решта – для виконання обчислень), що працюють на частоті 1,6 ГГц. Графічні процесори в суперкомп'ютері не використовуються. Разом в системі задіяно 98304 процесорів і, відповідно, 1572864 обчислювальних ядер.

3.9. Галузі застосування суперкомп'ютерів

З огляду на обчислювальні та вартісні характеристики наведених вище комп'ютерів виникають такі запитання:

– які задачі є настільки важливими, що потребують використання комп'ютерів вартістю сотні мільйонів доларів?

– які задачі є настільки складними, що одного процесора недостатньо?

Може видатися, що із зростанням продуктивності персональних комп'ютерів

і робочих станцій, а також серверів, сама потреба у суперкомп'ютерах знижуватиметься. Однак, це не так. З одного боку, багато програм можуть тепер успішно виконуватися на робочих станціях, але, з іншого боку, час показав, що стійкою тенденцією є поява нових завдань, для яких необхідно використовувати суперкомп'ютери. Нижче наведено невеликий перелік галузей людської діяльності, де необхідно використовувати суперкомп'ютери:

– автомобілебудування;
– нафто- і газовидобуток;
– фармакологія;
– прогноз погоди і моделювання зміни клімату;
– сейсморозвідка;
– геофізичні задачі моделювання процесів Землі, океану, атмосфери;
– проектування електронних пристроїв;
– синтез нових матеріалів;
– динаміка рідин;
– теоретична і експериментальна фізика, зокрема ядерна фізика та фізика високих енергій, квантова хромодинаміка та інші;
– квантова хімія;
– молекулярна динаміка;
– структурна інженерія;
– інженерія хімічних процесів;
– генна інженерія;
– сортування та пошук у базах даних;
– криптографія та криптоаналіз;
– візуалізація та обробка зображень;
– обробка сигналів;
– збирання даних.

Для того, щоб оцінити складність розв'язуваних на практиці завдань, візьмемо конкретну предметну галузь, наприклад, оптимізацію процесу видобутку нафти. Отже, є підземний нафтовий резервуар з деякою кількістю пробурених свердловин. Однією з них на поверхню відкачують нафту, а іншими назад закачують воду. Потрібно змоделювати ситуацію у цьому резервуарі, щоб оцінити запаси нафти або зрозуміти необхідність у додаткових свердловинах.

Приймемо спрощену схему, за якою область, що моделюється, відображається в куб, проте і її буде достатньо, щоби оцінити кількість необхідних арифметичних операцій. Розміри куба, за яких можна отримувати правдоподібні результати – це 100 х 100 х 100 точок. У кожній точці куба треба обчислити від 5 до 20 функцій: три компоненти швидкості, тиск, температуру, концентрацію компонентів (вода, газ і нафта – це мінімальний набір компонентів, у реалістичніших моделях розглядають, наприклад, різні фракції нафти). Значення функцій знаходять, розв'язуючи нелінійні рівняння, що вимагає від 200 до 1000 арифметичних операцій. І нарешті, якщо досліджується нестаціонарний процес, тобто потрібно зрозуміти, як ця система поводитиметься в часі, то роблять 100-1000 кроків за часом. У результаті отримуємо 10^6 (точок сітки) х 10 (функцій) х 500 (операцій) х 500 (кроків за часом) = 2.5 х $Ю^{12}$, тобто 2500 млрд арифметичних операцій для виконання лише одного розрахунку без врахування зміни параметрів моделі та відстеження поточної ситуації за зміни вхідних даних. Такі розрахунки необхідно робити багато разів, що накладає дуже жорсткі вимоги на продуктивність використовуваних обчислювальних систем. Схожі приклади можна знайти всюди.

На рис. 3.30 подано задачі, для розв'язання яких необхідно застосовувати суперкомп'ютери, а також потрібні для їх розв'язання комп'ютерні ресурси.

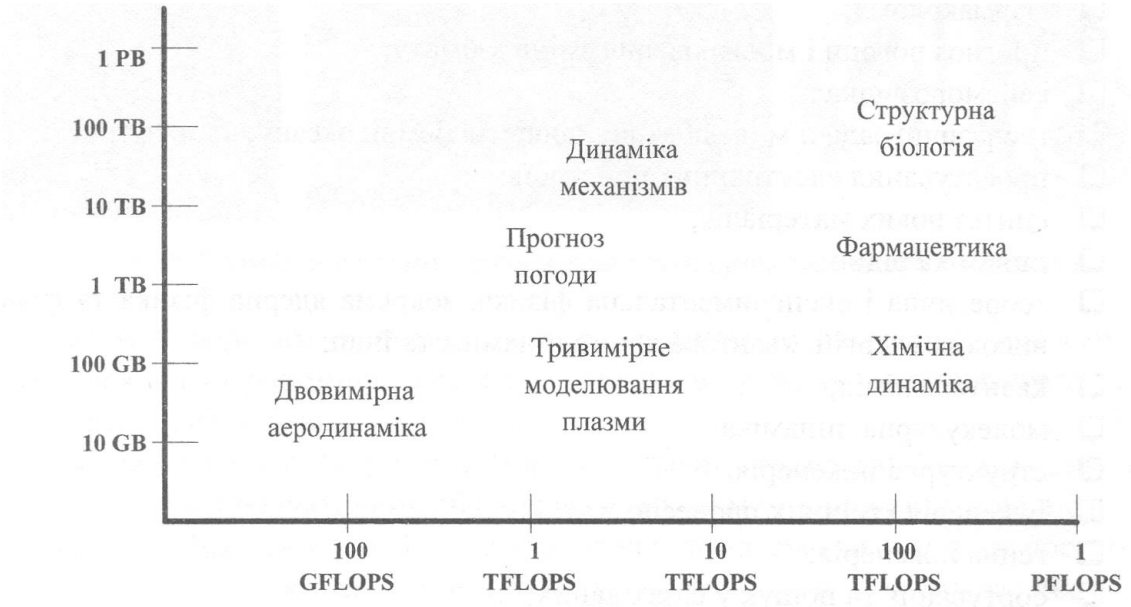


Рисунок 3.30. – Задачі та комп'ютерні ресурси

Видно, що ємність пам'яті досягає одного петабайту (1000 терабайтів, або 1000000 гігабайтів) за умови, що швидкодія має бути один PFLOPS (тобто 1000 TFLOPS, або 1000 000 GFLOPS). Зрозуміло, що межа необхідних комп'ютерних ресурсів є рухомою. Надати ресурси, які вимагаються наведеними задачами, за допомогою стандартних однопроцесорних систем неможливо. Це спричинює використання багатопроцесорних комп'ютерних систем як магістрального напрямку досягнення високої продуктивності.

3.10. Проблеми застосування суперкомп'ютерів

На жаль, надвисока продуктивність суперкомп'ютерів абсолютно компенсується складнощами їх використання. Розглянемо такий приклад. Якщо у вас є програма і доступ, скажімо, до 1000-процесорного комп'ютера, то ви цілком законно очікуєте, що програма буде виконуватися в 1000 разів швидше, ніж на одному процесорі. А цього, скоріш за все, не буде.

Припустимо, що у вашій програмі частка операцій, які потрібно виконувати послідовно, дорівнює *f,* де 0 </< 1 (при цьому частка визначається не статичним числом рядків коду, а кількістю операцій у процесі виконання). Крайні випадки в значеннях / відповідають повністю паралельним *(f*= 0) і повністю послідовним (/= 1) програмам. Для того, щоб оцінити, яке прискорення *S* можна отримати на комп'ютері з *p* процесорами за цього значення *f,* можна скористатися законом Амдала. Припустимо, що у вашій програмі лише 10 % послідовних операцій, тобто /=0,1. За законом Амдала, скільки б ви процесорів не використовували, прискорення роботи програми більш ніж у десять разів ніяк не отримаєте, причому прискорення у 10 разів – це теоретична верхня оцінка найкращого випадку, коли жодних інших негативних чинників немає.

Отже, перш ніж грунтовно переробляти код для переходу на паралельний комп'ютер (а будь-який суперкомп'ютер є таким), треба подумати над можливим результатом. Якщо, оцінивши закладений у програмі алгоритм, ви зрозуміли, що частка послідовних операцій велика, то на значне прискорення розраховувати не варто і потрібно думати про заміну окремих компонент алгоритму.

У ряді випадків послідовний характер алгоритму змінити не так складно. Припустимо, що в програмі є такий фрагмент для обчислення суми п чисел:

```
s = 0
Do i = 1, n
s = s + a (i)
EndDo
```

За своєю природою він строго послідовний, оскільки на *i*-й ітерації циклу потрібно мати результат (*i*-1)-ї ітерації, і всі ітерації виконуються одна за однією.

Маємо 100 % послідовних операцій, а отже, і жодного ефекту від використання паралельних комп'ютерів. Разом з тим, вихід очевидний. Оскільки в більшості реальних випадків немає істотної різниці, в якій послідовності додавати числа, виберемо іншу схему додавання. Спочатку знайдемо суми пар сусідніх елементів: a(1) + a(2), a(3) + a(4), a(5) + a(6) і т.д. Зауважимо, що за такою схемою всі пари можна додавати одночасно. На наступних кроках будемо діяти аналогічно, отримавши варіант паралельного алгоритму.

Крім особливостей алгоритму, необхідно врахувати й інші чинники, що можуть впливати на прискорення виконання алгоритму. Наприклад, доступні вам процесори можуть бути різнорідні за своєю продуктивністю, а отже, настає

такий момент, коли якийсь із них ще працює, а інший вже все зробив і марно простоює в очікуванні. Якщо розкид у продуктивності процесорів великий, то й ефективність всієї системи за рівномірного завантаженне процесорів буде вкрай низькою. Якщо припустити, що всі процесори однакові, настане момент, коли процесори виконали свою роботу, і результат потрібно буде передати іншим для продовження процесу додавання, а це знову вимагатиме часу, протягом якого процесори знову простоюватимуть.

Отже, змусити суперкомп'ютер працювати з максимальною ефективністю на конкретній програмі є складним завданням. Твердження: що потужніший комп'ютер, то швидше на ньому можна вирішити певне завдання, є справеливим не завжди, що можна пояснити простим побутовим прикладом. Якщо один землекоп вириє яму 1 м х 1 м х 1 м за 1 годину, то два такі землекопи це зроблять за ЗО хв – у це можна повірити. А за скільки часу цю роботу зроблять 60 землекопів? Невже за 1 хвилину? Звичайно ж ні. Починаючи з деякого моменту вони просто заважатимуть один одному, не прискорюючи, а сповільнюючи процес. Так само і в комп'ютерах: якщо задача занадто мала, то ми будемо довше займатися розподілом роботи, синхронізацією процесів, складанням результатів тощо, ніж безпосередньо корисною роботою.

3.11. Персональні суперкомп'ютери

Отже, суперкомп'ютери, які мають надвисоку продуктивність, використовують для виконання складних інженерних і наукових розрахунків та інших ресурсомістких завдань. Однак у них є серйозні недоліки, серед яких варто насамперед назвати величезну споживану потужність і дуже високу вартість розроблення та подальшої експлуатації. Крім того, однією з ключових проблем, що стоять перед вченими і дослідниками, є доступ до цих обчислювальних ресурсів. Іще більшою проблемою є забезпечення ефективного використання обчислювальної потужності, яку надає суперкомп'ютер.

Тому на сучасному етапі розвитку галузі високопродуктивних обчислювальних систем значну увагу приділяють створенню так званих "персональних суперкомп'ютерів", які могли б порівнятися за продуктивністю з кластерами суперкомп'ютерів, мали помірне енергоспоживання і невеликі габарити.

Термін "персональний суперкомп'ютер" з'явився недавно. У 2008 році з'явилася інформація про те, що фірма *Cray* представила персональний суперкомп'ютер CX1, що вмикався до стандартної мережі живлення і мав габарити, які дозволяли його розмістити під робочим столом.

У тому ж році компанія *NVIDIA* заявила про те, що їй вдалося "перетворити персональні суперобчислення на реальність". У новому продукті, що отримав назву *Tesla Personal Supercomputer,* використовують для обчислень графічні процесори і, за словами розробників, це забезпечує порівнянну з сучасними кластерами продуктивність за менших у сто разів вартості і

габаритів стандартних настільних робочих станцій. Анонс відеокарти *Tesla C1060,* яку покладено в основу суперкомп'ютера, відбувся влітку 2008 року.

Восени 2009 року компанія *Silicon Graphics International* поширила інформацію про свій персональний суперкомп'ютер *Octane III*, побудований на основі чоти- риядерних (серії *5500)* або шестиядерних (серії *5600*) мікропроцесорів *Intel Xeon* та графічних процесорів *NVIDIA Tesla CI060.*

Зупинимось детальніше на особливостях цих персональних суперкомп'ютерів.

3.11.1. Персональний суперкомп'ютер Tesla Personal Supercomputer фірми NVIDIA

Практично всі сучасні суперкомп'ютери побудовано на основі паралельно включених універсальних мікропроцесорів. Разом з тим, відомо, що за рахунок спеціалізації та апаратної адаптації обчислювальних засобів можна досягти прискорення виконання завдань на І-3 порядки. Тому існують спеціалізовані процесори, які з задачами, на які вони орієнтовані, переважають за продуктивністю не лише звичайні, а часто й суперкомп'ютери. З іншого боку, суперкомп'ютер повинен бути універсальним і забезпечувати необхідну продуктивність виконання довільного обчислювального завдання. Тому часто поєднують ці два підходи та створюють універсальні високопродуктивні системи на основі універсальних мікропроцесорів та спеціалізованих процесорів. За таким підходом отримують високі показники продуктивності систем, виконуючи завдання, на які орієнтований спеціалізований процесор. Наприклад, сучасні графічні процесори (*GPU – Graphics Processing Unit*) розвинулися до того рівня, коли безліч реальних завдань можна виконувати за їх допомогою набагато швидше, ніж за допомогою систем на основі багатоядерних універсальних мікропроцесорів. Тому і вирішили об'єднати в одній системі графічні процесори з багатоядерними центральними процесорами.

В основі персонального суперкомп'ютера *Tesla Personal Supercomputer* покладено потужну відеокарту *Tesla CI 060,* що підтримує технологію *GPGPU*-обчислень *NVIDIA CUDA*. *GPGPU* [5] (англ. *General-Purpose computing on Graphics Processing Units*) – це технологія використання графічного процесора відеокарти для виконання загальних обчислень, які зазвичай виконує центральний процесор. *CUDA* [6] (англ. *Compute Unified Device Architecture*) – це архітектура паралельних обчислень фірми *NVIDIA,* яка дає змогу програмістам реалізовувати обчислювальні алгоритми мовою програмування С та виконувати їх на графічних процесорах, що підтримують технологію *GPGPU.* При цьому програміст має доступ до набору інструкцій графічного процесора, керує його пам'яттю, виконуючи на ньому складні паралельні обчислення. Зовнішній вигляд персонального суперкомп'ютера *Tesla Personal Supercomputer* наведено на рис. 3.31.

Рисунок 3.31. – Зовнішній вигляд персонального суперкомп'ютера *Tesla Personal Supercomputer*

Суперкомп'ютер може містити три або чотири відеокарти *Tesla CI060* і в максимальній комплектації об'єднує 960 паралельно працюючих обчислювальних ядер (при встановлених чотирьох відеокартах) та забезпечує продуктивність 3,732 TFLOPS. Основні характеристики відеокарти *Tesla C1060* наведено нижче [7]:

– Кількість потокових процесорних ядер: 240.
– Тактова частота ядер: 1,296 ГГц.
– Продуктивність виконання операцій з рухомою комою над даними у форматі з одинарною точністю: 933 GFLOPS.
– Продуктивність виконання операцій з рухомою комою над даними у форматі з подвійною точністю: 78 GFLOPS.
– Ємність пам'яті: 4 ГБ.
– Пропускна спроможність пам'яті: 102 ГБ/с.
– Інтерфейс: PCI Express 2.0x16.
– Споживана потужність: 160 Вт (пікова 200 Вт).

Суперкомп'ютер підтримує платформи *Microsoft Windows XP* (64-бітову та 32-бітову), *Linux* 64-бітову та 32-бітову (рекомендовано 64-бітову), *Red Hat Enterprise Linux 4* і 5, *SUSE* 10.1, 10.2 та 10.3.

3.11.2. Персональний суперкомп'ютер CX1 фірми CRAY

Персональний суперкомп'ютер *CX1* фірми *CRAY* побудований як шасі з вісьмома слотами, у які можна встановити до восьми плат (вузлів). Конфігурацію цього суперкомп'ютера підбирають за потребою замовника, комбінуючи вузли декількох типів. Вузли є одно- та двороз'ємними і основуються на двоядерних або чотириядерних процесорах *Intel Xeon.* На одноroз'ємному вузлі розміщують до 16 таких процесорів, отримуючи так до 64 паралельно працюючих ядер [8].

Зовнішній вигляд персонального суперкомп'ютера CX1 показано на рис. 3.32.

Рисунок 3.32. – Зовнішній вигляд персонального суперкомп'ютера *CX1* фірми *Cray*

Пропонується чотири основні типи вузлів: обчислювальний вузол (*Compute Blade),* вузол візуалізації (*Visualization Blade),* вузол *GPGPU (GPGPU Blade)* та вузол зберігання даних (*Storage Blade).* Типи вузлів із позначеннями їх моделей наведено на рис. 3.33.

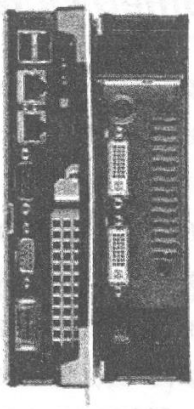
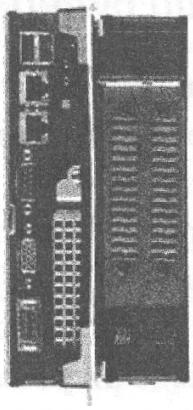
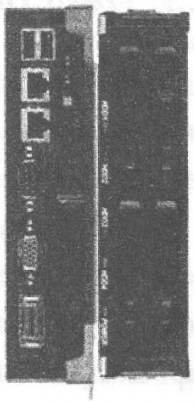
CN5500	*CV5501*	*CT5501*	*CS5504*
			
Обчислювальний вузол	Вузол візуалізації	Вузол *GPGPU*	Вузол зберігання даних

Рисунок 3.33. – Типи вузлів персонального суперкомп'ютера *CX1* фірми *Cray*

У табл. 3.2 наведено основні характеристики цих вузлів.

Таблиця 3.2.

Основні характеристики вузлів персонального суперкомп'ютера CX1 фірми Cray

Тип вузла	**CN5500 Обчислювальний вузол**	**CV5501 Вузол візуалізації**	**CT5501 Вузол GPGPU**	**CS5504/CS5508 Вузол зберігання даних**
Кількість займаних слотів	1 слот	2 слоти		
Процесор	2 x Intel Xeon 55XX			
Оперативна пам'ять	48 ГБ (за іншими даними 64 ГБ)	96 ГБ (за іншими даними 128 ГБ)		
Зовнішня пам'ять	1 *SATA/SSD* (ємність змінна, за даними різних джерел від 80 до 320 ГБ)			4/8 *SATA/SSD* (2/4 ТБ)
Інтерфейси розширення	*PCI Express* 2.0x16	*PCI Express* 2.0x16 (зарезервований для графічної карти)	*PCI Express* 2.0x16 (зарезервований для *GPGPU)*	*PCI Express* 2.0x16
Зовнішні інтерфейси	*Gigabit Ethernet* та *InfiniBand*			

Обчислювальні задачі можна виконувати на кожному вузлі окремо або розподіляти за вузлами комп'ютера. Передбачено можливість нарощення потужності шляхом об'єднання двох або трьох таких комп'ютерів в одну систему без додаткового обладнання. Подальше нарощення можливе з використанням додаткових свічів.

3.11.3. Персональний суперкомп'ютер Octane III фірми SGI

Персональний суперкомп'ютер *Octane III* може містити від двох до десяти автономних комірок. Зовнішній вигляд персонального суперкомп'ютера *Octane III* показано на рис. 3.34.

Архітектура комірки залежить від моделі суперкомп'ютера. *SGI* пропонує дві моделі суперкомп'ютерів: *OC3-10TY9* (кластерне робоче місце – *Deskside Cluster),* до складу якої може входити до десяти автономних комірок, та *OC3-2TY12* (графічна робоча станція – *Graphics Workstation),* яка містить дві автономні комірки. Характеристики комірки наведено нижче [9]:

- Кількість універсальних процесорів: 2, *IntelXeon* серії *5500* або *5600;*
- Кількість потокових процесорних ядер: 8 або 12, залежно від серії процесорів;
- Продуктивність в операціях з рухомою комою: 726 Гфлопс;
- Ємність пам'яті: 96 Гб;
- Зовнішній інтерфейс: *Gigabit Ethernet* та *InfiniBand.*

Рисунок 3.34 – Зовнішній вигляд персонального суперкомп'ютера *Octane III* фірми *SGi*

Споживана потужність суперкомп'ютера становить 2000 Вт. Цікаво, що автономна комірка суперкомп'ютера моделі *OC3-10TY9* побудована винятково на основі універсальних мікропроцесорів *Intel Xeon,* а комірка моделі *OC3-2TY12* додатково містить дві відеокарти *NVIDIA Tesla CI 060,* що використовуються як графічні процесори загального призначення. Кожну з них під'єднано через окремий інтерфейс розширення *PCI Express* 2.0x16.

До переліку підтримуваних операційних систем входять *Microsoft Windows HPC Server 2008, SUSE Linux Enterprise Server* та *Red Hat Enterprise Linux.* Конфігурації на основі *Linux* постачаються в комплекті зі системним програмним забезпеченням *SGI ProPack, SGI Isle Cluster Manager* та *Altair PBS Professional Scheduler,* що дає змогу додатково прискорити роботу суперкомп'ютера.

3.12. Підходи до побудови суперкомп'ютерів

За наведеним вище описом сучасних високопродуктивних обчислювальних систем можна зробити висновок, що побудовані вони за такими принципами:

– Найсучасніший технологічний рівень.

– Використання великої кількості передових універсальних процесорів.

– Використання великих об'ємів оперативної і зовнішньої пам'яті.

– Розпаралелювання завдань і паралельна обробка даних.

Підвищують продуктивність таких комп'ютерів переважно нарощуванням обчислювальних потужностей їхніх вузлів: процесорів, об'ємів

пам'яті, і покращенням характеристик, що відбувається завдяки розвитку інтегральних технологій загалом. У деяких випадках підвищення продуктивності досягають, використовуючи спеціалізовані обчислювальні модулі на зразок графічних процесорів. Необхідно зазначити, що такий підхід дає можливість прискорити виконання далеко не всіх завдань, а лише тих, для виконання яких призначені спеціалізовані обчислювальні модулі.

Персональні суперкомп'ютери як останнє досягнення розвитку високопродуктивних обчислювальних систем з'явилися останніми роками завдяки стрімкому розвитку мікроелектронного виробництва. Звичайно, вони не можуть зрівнятися за продуктивністю з традиційними суперкомп'ютерами, однак мають низку переваг за надзвичайно важливими характеристиками, такими як доступність для користувача, вартість виготовлення та експлуатації, споживана потужність.

Зважаючи на те, що сучасні універсальні процесори є багатоядерними і паралельними, поява персональних суперкомп'ютерів є закономірною. Разом з тим необхідно зазначити, що майже всі сучасні персональні суперкомп'ютери побудовано на основі універсальних мікропроцесорів та спеціалізованих обчислювальних модулів, що дає змогу значно підвищити їх загальну продуктивність. Ми вже згадували, що завдяки спеціалізації та апаратної адаптації обчислювальних засобів можна прискорити виконання завдань на 1-3 порядки. Саме такий підхід дає можливість досягти максимальної продуктивності та ефективності використання обладнання, причому спеціалізовані обчислювальні засоби позбавлені надлиш- ковості. З іншого боку, суперкомп'ютер повинен бути універсальним і забезпечувати необхідну продуктивність виконання довільного обчислювального завдання.

Постає запитання: як цього досягти, тобто як поєднати універсальні мікропроцесори та апаратну орієнтацію на виконувані алгоритми? Відповіддю на це запитання є використання альтернативного відносно розглянутих підходу до побудови універсальних високопродуктивних систем – на основі універсальних мікропроцесорів та реконфігуровних процесорних модулів. Реконфігуровні процесорні модулі, або, як їх називають, прискорювачі, апаратно реалізують заданий обчислювальний алгоритм, тобто є спеціалізованими обчислювальними пристроями, які дають змогу змінювати конфігурацію завдяки тому, що реалізуються в кристалах програмовної логіки. Звичайно, спеціалізовані процесори, реалізовані в реконфігуровних кристалах, поступаються за продуктивністю замовним спеціалізованим процесорам, однак підхід фактично дає можливість будувати високопродуктивні обчислювальні системи на основі універсальних мікропроцесорів та спеціалізованих процесорних модулів, і, що найважливіше, отримувати високі показники продуктивності таких систем для виконання довільних завдань, оскільки в реконфігуровному прискорювачі можна реалізувати довільний спеціалізований процесор.

ЛЕКЦІЯ 4. КОРОТКА ХАРАКТЕРИСТИКА РЕЙТИНГУ TOP500. СУЧАСНІ ТЕНДЕНЦІЇ РОЗВИТКУ ПРОЦЕСОРІВ. ГІБРИДНІ ВИСОКОПРОДУКТИВНІ ОБЧИСЛЮВАЛЬНІ СИСТЕМИ. ОРГАНІЗАЦІЯ МІЖПРОЦЕСОРНИХ ЗВ'ЯЗКІВ – КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ. ХАРАКТЕРИСТИКИ ІНТЕРКОНЕКТУ. ПОБУДОВА КЛАСТЕРІВ, БАГАТОПРОЦЕСОРНИХ СЕРЕДОВИЩ ТЕЛЕКОМУТАЦІЙНИХ МЕРЕЖ ДЛЯ РОЗПОДІЛЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМ.

4.1. Коротка характеристика рейтингу TOP500

TOP500 – проект зі складання рейтингу і описів 500 найпотужніших суспільно відомих комп'ютерних систем світу. Проект був запущений в 1993 році і публікує актуальний список суперкомп'ютерів двічі на рік (у червні і листопаді). Цей проект спрямований на забезпечення надійної основи для виявлення та відстеження тенденцій в області високопродуктивних обчислень. Основою для рейтингу є результати виконання тесту LINPACK (HPL), вирішального великі СЛАР.

На початку 1990 -х років виникла необхідність отримання порівняльних характеристик і метрик суперкомп'ютерів. Після рейтингів 1986-1992 років з метриками, заснованими на кількості векторних процесорів, в університеті Мангейма у Erich Strohmaier і Hans Werner Meuer виникла ідея почати щорічно порівнювати суперкомп'ютери за допомогою єдиної методики.

На початку 1993 року Джек Донгарра (англ.) взяв участь у цьому проекті зі своїм тестом Linpack. Перша версія списку була готова в травні 1993 року. Вона частково була заснована на даних, доступних в мережі, включаючи дані джерела:

- Статистика з суперкомп'ютерів Мангейма.
- Список найпотужніших світових обчислювальних вузлів, оновлюваний Гюнтером Арендтом.
- Інформація від Девіда Кехнер.

Інформація з цих джерел використовувалася для створення перших двох списків TOP500.

З червня 1993 TOP500 складається двічі на рік і грунтується тільки на інформації від організацій, в яких встановлені комп'ютери, і від виробників. Запит інформації розсилається за 8 тижнів до виходу нової версії списку.

У 2013 році Джек Донгарра запропонував новий тест для ранжирування суперкомп'ютерів, High Performance Conjugate Gradient (HPCG).

Критика методики включає в себе думки, що Linpack не відповідає реальній завантаженні комп'ютерів. Крім того, деякі організації відмовляються виділяти весь суперкомп'ютер на час, необхідний для запуску тесту. При цьому із зростанням продуктивності кластерів для запуску Linpack потрібно все більше часу, аж до декількох діб.

На деяких суперкомп'ютерах, які могли б потрапити в рейтинг, тест linpack або не запускають, або не повідомляють його результати.

Один з авторів методики складання списку Hans Werner Meuer відзначав, що через невірне розуміння результатів TOP500 політики намагаються використовувати рейтинг як універсальний. Водночас список відображає виключно можливості систем за рішенням СЛАР і ніяк не пов'язаний з продуктивністю на інших типах завдань.

Таблиця 4.1.

TOP500 (2013 рік)

№ з/п	Rmax Rpeak (Pflops)	Назва	Архітектура Тип процесора, мережа	Виробник	Місце знаходження, країна, рік	Операційна система
1	33.863 54.902	*Tianhe-2*	NUDT Xeon E5–2692 + Xeon Phi, Custom	NUDT	Національний комп'ютерний центр в Гуанчжоу , 2013	Linux (SLURM)
2	17.590 27.113	*Titan*	Cray XK7 Opteron 6274 + Tesla K20X, Custom	Cray	Національна лабораторія Оук-Ридж (ORNL) Тенесі , 2012	Linux (CLE, SLES based)
3	17.173 20.133	*Sequoia*	Blue Gene/Q PowerPC A2, Custom	IBM	Ліверморська національна лабораторія , 2013	Linux (RHEL and CNK)
4	10.510 11.280	*K computer*	RIKEN SPARC64 VIIIfx, Tofu	Fujitsu	RIKEN Японія, 2011	Linux
5	8.586 10.066	*Mira*	Blue Gene/Q PowerPC A2, Custom	IBM	Аргонська національна лабораторія , 2013	Linux (RHEL and CNK)
6	6.271 7.788	*Piz Daint*	Cray XC30 Xeon E5-2670, NVIDIA K20x	Cray	Swiss National Supercomputing Centre (CSCS) , 2013	Cray Linux Environment
7	5.168 8.520	*Stampede*	PowerEdge C8220 Xeon E5–2680, Infiniband	Dell	Texas Advanced Computing Center США, 2013	Linux

Продовження таблиці 4.1

№ з/п	Rmax Rpeak (Pflops)	Назва	Архітектура Тип процесора, мережа	Виробник	Місце знаходження, країна, рік	Операційна система
8	5.008 5.872	*JUQUEEN*	Blue Gene/Q PowerPC A2, Custom	IBM	Дослідницький центр Юліх, 2013	Linux (RHEL and CNK)
9	4.293 5.033	*Vulcan*	Blue Gene/Q PowerPC A2, Custom	IBM	Ліверморська національна лабораторія, 2013	Linux (RHEL and CNK)
10	2.897 3.185	*SuperMUC*	iDataPlex DX360M4 Xeon E5–2680, Infiniband	IBM	Leibniz-Rechenzentrum Німеччина, 2012	Linux

Результати Linpack також не можуть безпосередньо допомогти у виборі комп'ютера для організації. Hans Meuer зазначив користь існування пакетів, що складаються з декількох різнорідних тестів, зокрема, «HPC Challenge Benchmark» (7 тестів, що вимірюють продуктивність різних підсистем суперкомп'ютера).

– Rmax – Найвищий результат, отриманий при використанні системи тестів Linpack (реалізація HPL). Це число використовується для порівняння швидкодії комп'ютерів. Вимірюється в TFLOPS.

– Rpeak – Теоретична пікова продуктивність системи. Вимірюється в TFLOPS.

Системи, які посіли перше місце з 1993:

– Inspur Тяньхе-2 (КНР, червень 2013 – дотепер).
– Cray Titan (США, листопад 2012 – червень 2013).
– IBM Sequoia Blue Gene/Q (США, липень 2012 – листопад 2012).
– Fujitsu K computer (Японія, липень 2011 – липень 2012).
– NUDT Tianhe-1A (КНР, листопад 2010 – липень 2011).
– Cray Jaguar (США, листопад 2009 – листопад 2010).
– IBM Roadrunner (США, липень 2008 – листопад 2009).
– IBM Blue Gene/L (США, листопад 2004 – липень 2008).
– NEC Earth Simulator (Японія, липень 2002 – листопад 2004).
– IBM ASCI White (США, листопад 2000 – липень 2002).
– Intel ASCI Red (США, липень 1997 – листопад 2000).
– Hitachi CP-PACS (Японія, листопад 1996 – липень 1997).
– Hitachi SR2201 (Японія, липень 1996 – November 1996).
– Fujitsu Numerical Wind Tunnel (Японія, листопад 1994 – липень 1996).

– Intel Paragon XP/S140 (США, липень 1994 – листопад 1994).

– Fujitsu Numerical Wind Tunnel (Японія, листопад 1993 – липень 1994).

– TMC CM-5 (США, липень 1993 – листопад 1993).

4.2. Багатоядерні процесори

4.2.1. Багатоядерні мікропроцесори. Закон Мура для ядер

Розвиток мікропроцесорної техніки в області універсальних мікропроцесорів йде шляхом постійного підвищення їх продуктивності. Традиційними напрямами такого розвитку є:

– Підвищення тактової частоти роботи МП, яке в основному забезпечуються шляхом збільшення кількості щаблів у конвеєрі, що призводить до великих втрат часу при необхідності перезавантаження конвеєра внаслідок залежності команд за даними і конфліктів з управлінням. Крім того, при довгих конвеєрах час передачі даних з етапу на етап стає порівнянняльним з часом виконання етапу.

– Збільшення кількості одночасно виконуваних команд за рахунок збільшення числа конвеєрів в МП. Це занадто ускладнює управління МП, та й число команд в програмі, які можна виконати одночасно, невелике.

У зв'язку з успіхами технології в галузі мікроелектроніки з'явився третій шлях. На сьогодні зберігається експоненціальне зростання числа транзисторів (малюнок нижче), який описується експериментальним законом Мура [9].

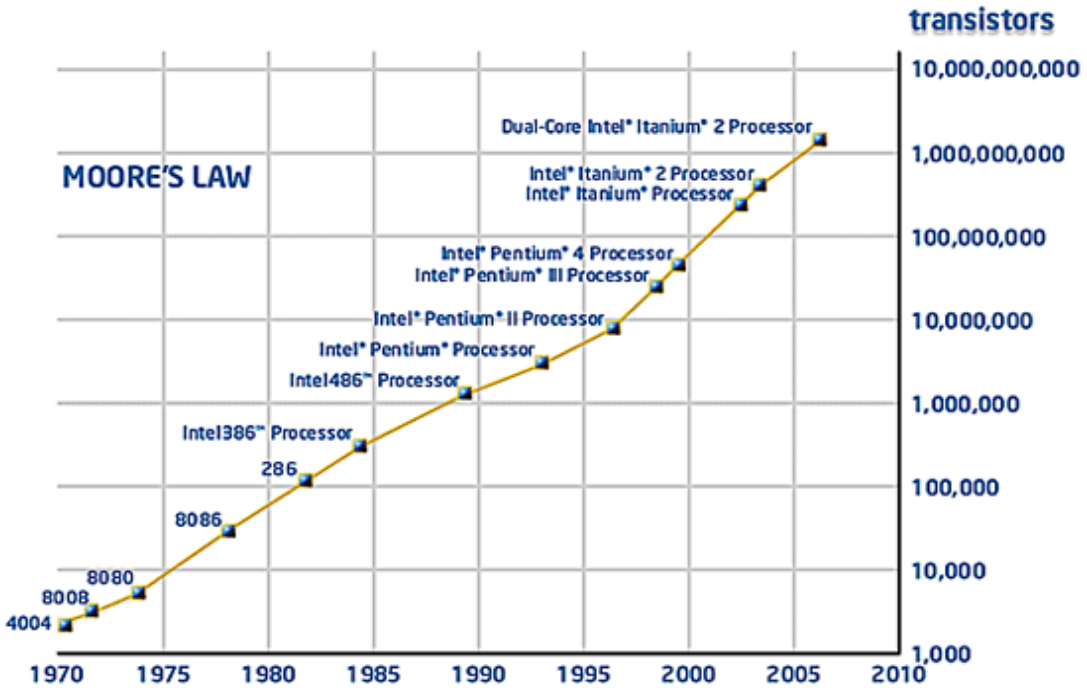


Рисунок 4.1. – Зростання числа транзисторів

Ця кількість транзисторів на кристалі дозволяє будувати на одному кристалі велику кількість закінчених процесорів, які називають багатоядерними.

Багатоядерний процесор (БЯП) – це центральний процесор, що містить два і більше обчислювальних ядра на одному процесорному кристалі. Кожне ядро за функціональними і технічними можливостями може відповідати центральному або спеціалізованому процесору. Ядра одного багатоядерного процесора можуть бути однаковими або відрізнятися за своїми можливостями. Основні відомості по БЯП розташовані за адресою [20].

Закон Мура, що визначає зростання числа транзисторів на кристалі, формулюється таким чином:

Кількість транзисторів на кристалі подвоюється кожні два роки.

Мур вказав також, що результатом зростання числа елементів на кристалі стане зниження вартості на елемент. При цьому, темпи збільшення числа елементів (і функцій, відповідно) – вище, ніж темпи зростання вартості виробництва кристала. Це і є рушійною силою розвитку багатоядерності.

Дяконов [21] записав закон Мура в математичній формі:

$$K = K_0 2^{\frac{t-t_0}{2}} = K_0 2^{\frac{\Delta t}{2}} \tag{4.1}$$

де K, K_0, t, t_0 – кількість транзисторів початкова і через час Δt. Якщо взяти приклад мікропроцесора *i*486 з параметрами t_0 – 1990, K_0=10^6, то до 2010 отримуємо:

$$K = 10^6 2^{\frac{20}{2}} = 10^9, \tag{4.2}$$

що приблизно відповідає таблиці.

Якщо припустити, що на одне ядро витрачається 10^7 транзисторів, то отримуємо 100 ядер, що відповідає дійсності (48, 80 ядер від Intel). Таким чином, можна вважати, що закон Мура поширюється і на багатоядерні процесори:

Кількість ядер на кристалі подвоюється кожні два роки.

За цим законом слід очікувати до 2020 року 1000 ядер на одному кристалі. Слід уточнити поняття ядра:

– Згадуваний у розділі 1 суперкомп'ютер СКІФ-Аврора має 2048 ядер, але це не ядра одного кристала, а сумарне число ядер всіх 4-ядерних Інтел, використаних у цьому суперкомп'ютері.

– Фахівці з Токійського університету створили процесор, який складається з 512 ядер, розташованих на одному кристалі Кожне з 512 ядер відповідає за окрему математичну операцію. Цей кристал з частотою 500 мегагерц з'єднаний з картою PCI-X і здійснює підтримку центрального

процесора. По суті це не ядра, а спеціалізовані АЛП. Ядром ж слід вважати повноцінний процесор.

Багатоядерні процесори можуть використовуватися подвійно:

– Як автономні пристрої: ПЕОМ, робочі станції, сервери та ін У цьому випадку критерієм використання БЯ є збільшення номінального швидкодії БЯ кристала при різкому зниженні вартості одного ядра.

– Як елементна база обчислювальних кластерів. Тут критерій інший – максимальну швидкодію кристала при жорсткому обмеженні на споживану потужність і нагрів (це різні речі). У цьому випадку кількість ядер не обмежується – чим більше, тим краще.

Однак на цьому шляху виникає наступна перешкода – нагрів кристала, що потребують особливих методів охолодження кристала, а в разі багатокристальних систем у великих кластерах призведе до мегаватного споживання енергії.

На рис. 4.2 зліва представлений звичайний кремнієвий транзистор.

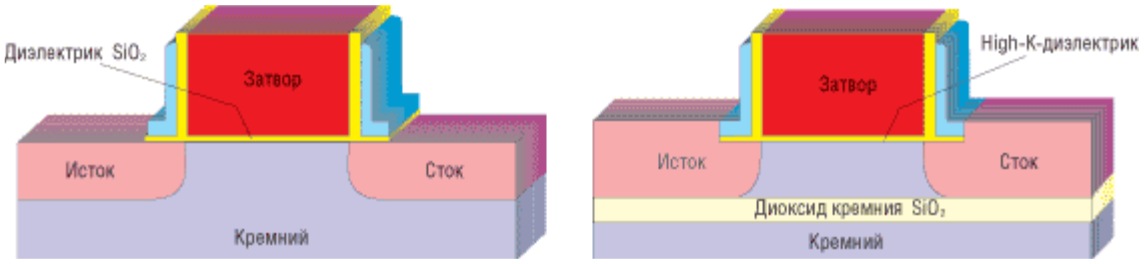


Рисунок 4.2. – Звичайний транзистор (ліворуч) і транзистор з ізоляцією проти витоків (праворуч)

При зменшенні розмірів такого транзистора в області нанорозмірів (нижче 100 нм) в m раз параметри транзистора і кристал, на якому він розташований, змінюються, як зазначено в таблиці [22]:

Таблиця 4.2.

Зміна параметрів транзистора

Характеристика	**Коефіцієнт**
Довжина затвору	1/m
Напруга	1/m
Щільність розміщення	m^2
Швидкість	m
Розсіююча потужність	$1/m^2$

– в m^2 разів зростає кількість транзисторів на кристалі, що в ідеалі збільшує в m^2 раз швидкодію БЯ за рахунок збільшення числа ядер.

– крім того, в m разів зменшується відстань між витоком і стоком, тому в m раз зростає швидкість перемикання транзистора, тобто частота. Отже, при збільшенні m зростання швидкодії кристала має порядок $V_{кр}=m^2$‘m=m^3.

З таблиці випливає, що розсіювана кристалом дорівнює:

Щільність розміщення ‘ Потужність транзистора = m^2‘1/m^2 = const

Це означає, що при зростанні транзисторів (ядер) на кристалі нагрів кристала не змінюється. При цьому кристал розсіює близько 100 вт, для чого потрібно стандартний радіатор і вентилятор. Це ідеальна ситуація.

Більш того, якщо врахувати, що частота $f=k1\times m$, а кількість процесорів $p=k2\times m\times m$, то швидкодія обчислювальних пристроїв зростає так:

$$V = f \times p = k1 \times m \times k2 \times m \times m = k \times m^3(t), \tag{4.3}$$

тобто по роках швидкість обчислень зростає кубічно, причому, більшою мірою за рахунок числа ядер, а не їх частоти.

На жаль, зменшення лінійних розмірів транзистора до 30 – 10 нм призводить до виникнення струмів витоку внаслідок тунельного ефекту. Струми витоку виникають (рис. 6.1 зліва):

– через шар діелектрика, що відокремлює область затвора від підкладки;

– між витоком і стоком при «вимкненому» стані транзистора.

Ці струми співставні з робочими струмами транзистора, і при збільшенні числа транзисторів на кристалі, він буде сильно перегріватися. Якщо не вживати заходів, це буде істотно обмежувати зростання числа транзисторів і ядер.

Використання нових ізолюючих матеріалів (рис.4.3 праворуч), як вважають фахівці, продовжує дію закону Мура до 2000 – го року, зберігаючи розсіюючу кристалом потужність до 100 Вт.

Однак, цих заходів надалі може виявитися недостатньо. Тоді застосовують додаткові заходи, наприклад: міжшарове водяне охолодження у кристала типу "сендвіч", виборче відключення незавантажених ядер і вузлів, зниження тактової частоти.

Переломним при переході від мікропроцесорів до багатоядерних кристалів виявився 2007 рік [23], але відбувалося це не миттєво, а було розтягнутого в часі.

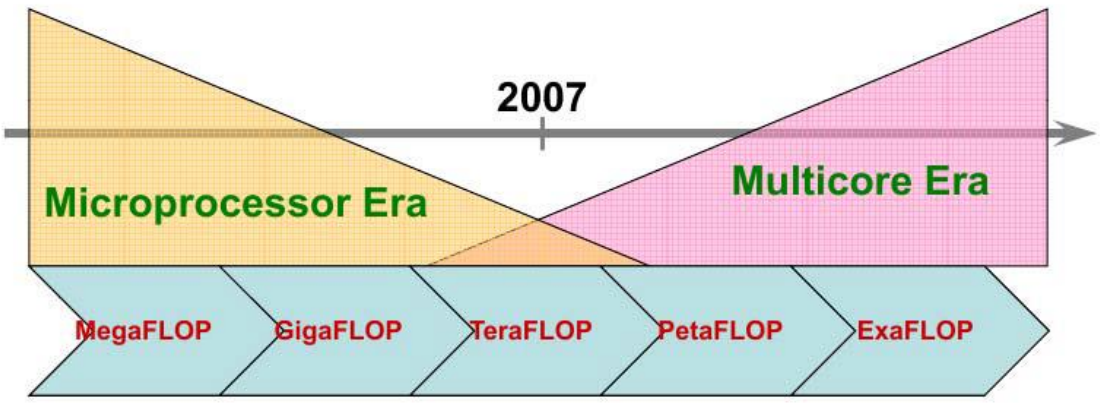


Рисунок 4.3. – Перехід від мікропроцесорів до багатоядерних кристалів

4.2.2. Дві архітектури багатоядерних процесорів

В області БЯП слід розрізняти поняття: архітектура БЯП і мікроархітектура ядра. Під мікроархітектурою ядра розуміють склад арифметико-логічних блоків (можуть бути співпроцесори VLIW, MMX), блоків звернення до пам'яті та реалізації умовних переходів. Архітектура ж визначає зв'язки ядер між собою (загальна або індивідуальна пам'ять), з пам'яттю, типи комутаторів, зв'язок із зовнішнім світом.

Далі будуть розглянуті два типи багатоядерних процесорів:

– Nehalem – 8-ядерний процесор із загальною пам'яттю для всіх ядер.

– Polaris – 80-ядерний експериментальний процесор з індивідуальною пам'яттю для кожного ядра.

У літературу ці два типи багатопроцесорних систем мають назви Multicore і Manycore. Різниця між цими двома архітектурами досить очевидна.

Архітектура з загальною пам'яттю:

– Полегшує програмування, оскільки не треба описувати переміщення даних між ядрами, дані знаходяться в загальній для всіх ядер пам'яті.

– Використання існуючого програмного забезпечення.

– Висока надійність системи, оскільки число ядер невелике.

– Головний недолік-число ядер невелика з-за обмежень спільної пам'яті. Система кешування згладжує цей фактор.

Архітектура з індивідуальною пам'яттю для кожного ядра використовує малі ядра з короткими конвеєрами і невеликими частотами. Але, малі ядра мають менше енергоспоживання, на кристалі їх може бути значно більше, вони забезпечують найкращу обчислювальну ефективність на ват.

4.2.3. Процесор Nehalem

Nehalem – однокристальний процесор з архітектурою загальної (що розділяється) пам'яті. Він має наступні технічні характеристики:

– Рік випуску 2008 – 2009.

– Процесори містять до 8 ядер.

– 45-нм технологічний процес.

– Тактова частота 3 ГГц.

– 2,3 млрд транзисторів.

– Тепловиділення 95 – 130 Вт.

– Ядро одночасно може оброблятися 2 потоки.

– У Nehalem кожен процесор має власну ОС. Це відрізняє його від загальноприйнятого розуміння SMP систем.

Мікроархітектура ядра. Ядро Nehalem виконує стандартну для мікропроцесорів послідовність кроків, показану на малюнку.

У ядрі використовуються 3 рівні кеш-пам'яті:

– 32 KB L1 – кеш для програм і 32 KB L1 кеш для даних на одне ядро.

– 256 KB L2 – кеш на ядро.

– до 24 MB L3 – кеш для всіх ядер.

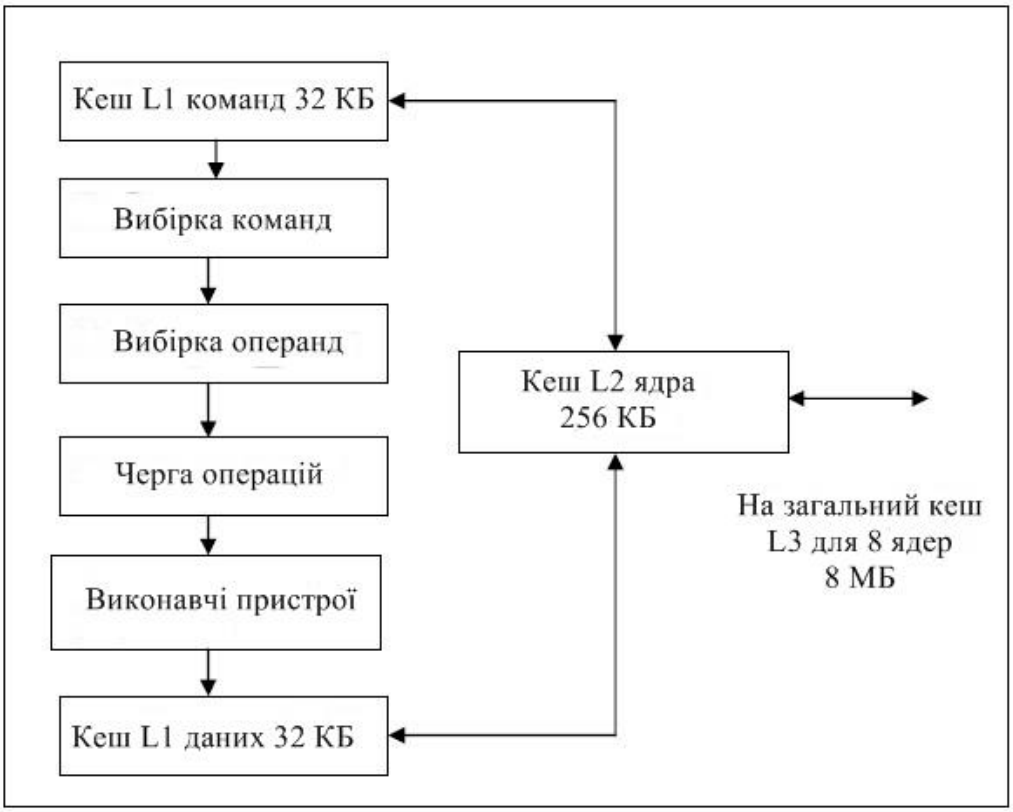


Рисунок 4.4. – Мікроархітектура ядра

*Склад виконавчих пристроїв одного ядра***.** Істотно попрацювавши над попередніми стадіями конвеєра Nehalem, інженери Intel залишили виконавчі пристрої нового процесора без помітних змін. Блок виконавчих пристроїв містить 6 портів. Порти 0,1 управляють роботою 8 арифметичних пристроїв для виконання цілочисельних та операцій з плаваючою крапкою. Порти 2-4 забезпечують роботу пам'яті, а порт 5 виконує деякі арифметичні операції і управляє операціями переходів. Отже, в Nehalem може одночасно виконуватися 6 операцій. Крім скалярних операцій ядро може виконувати команди SSE (або MMX) і потоки для Hyper-Threading. Технологія SSE дозволяла подолати 2 основні проблеми MMX – при використанні MMX неможливо було одночасно використовувати інструкції співпроцесора, так як його регістри використовувалися для MMX і для роботи з речовими числами. SSE включає в архітектуру процесора вісім 128-бітових регістрів, кожен з яких трактується як 4 послідовних значення з плаваючою крапкою одинарної точності. РЄ включає в себе набір інструкцій, який виробляє операції зі скалярними і упакованими типами даних.

Архітектура кристала. Процесор Nehalem має по чотири широкосмугових шини QuickPath. Рішення, вибране Intel під назвою QuickPath Interconnect (QPI), не є чимось новим; воно являє собою вбудований контроллер пам'яті і дуже швидку послідовну шину "точка-точка".

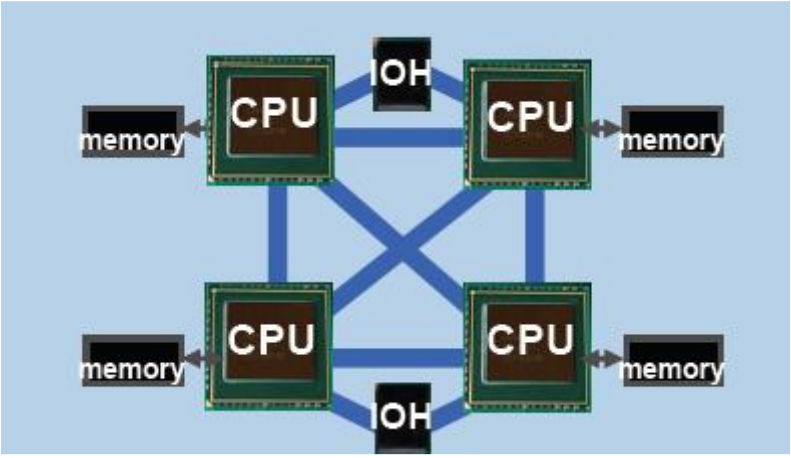


Рисунок 4.5. – Архітектура кристалу

З технічної точки зору інтерфейс QPI є двонаправленим з двома 20-бітними шинами, по одній на кожен напрямок, з яких 16 зарезервовано під дані, а решту чотири – під функції виправлення помилок або службову інформацію протоколу.

Це дозволяє:

– створювати повнозв'язні чотирипроцесорні системи, коли кожен процесор може отримувати доступ до будь-якій області пам'яті через один хоп QPI, оскільки кожен процесор безпосередньо підключений до трьох

– дозволяє будувати системи з декількома процесорними сокетами, здатними обробляти одночасно до 128 (8 ‘ 8 ‘ 2) процесів без використання додаткових пристроїв.

IOH – хаб – для зв'язку із зовнішнім світом, центральним процесором або іншими кристалами Nehalem.

Програмування. Система програмування Posex для потоків є низькорівневим інструментом програмування, тому вимагає великих знань і зусиль. Інтерфейс OpenMP є стандартом для програмування на масштабованих SMP-системах з роздільною пам'яттю, зокрема, для багатоядерних процесорів з роздільною пам'яттю. OpenMP ідеально підходить для розпаралелювання програм великими паралельними циклами. Розробник не створює нову паралельну програму, а просто додає в текст послідовної програми OpenMP-директиви.

Передбачається, що OpenMP-програма на однопроцессорной платформі може бути використана в якості послідовної програми, тобто немає необхідності підтримувати послідовну і паралельну версії. Директиви OpenMP просто ігноруються послідовним компілятором.

Що ж до Hyper-Threading (HT), то тут справа йде таким чином. Один процес спочатку розрахований на виконання тільки одного потоку команд (один СчАК). У нього є тільки один регістр-покажчик вибірки команд і один набір регістрів загального призначення. Процесор не може фізично виконувати декілька потоків одночасно.

Процес (process) – це деяка частина (одиниця) роботи, створювана операційною системою. Програма може складатися з декількох процесів.

Процес задається адресним простором та ідентифікатором. Адресний простір процесу ділиться на три логічних розділу: текстовий (код програми), інформаційний (дані) і стековий (для стеків програми).

Процес може містити кілька потоків. Різниця між потоками і процесами полягає в тому, що кожен процес має власний адресний простір, а потоки – ні. Ресурси, відкриті батьківським процесом, негайно стають доступними всім потокам.

Використання потоків має ряд переваг:

– Для перемикання контексту потрібно менше системних ресурсів.

– Досягається більш висока продуктивність програми.

– Для забезпечення взаємодії між завданнями не потрібно ніякого спеціального механізму.

– Програма має більш просту структуру.

– Перемикання процесора на потік мінімізовано аж до операцій збереження / відновлення цих покажчиків.

На жаль, багатопотокове паралельне програмування вимагає підвищених зусиль від програміста, що стосуються забезпечення коректності паралельної програми (однозначності результату її виконання незалежно від тимчасових характеристик індукованих потоків і використання загальних змінних), її ефективності (часте "зіткнення" потоків при зверненні до загальних програмних і апаратних ресурсів).

Потоки живуть тільки через простої пам'яті, блоків АЛУ та ін Розглянемо приклад. Якщо обидва потоки містять мало обчислень і активно звертаються до пам'яті, через єдиного інтерфейсу до пам'яті виконання потоків перетворюється практично на послідовне (неможливо прискорити операцію memcpy (), розділивши її на потоки).

Сфера застосування багатоядерних процесорів із загальною пам'яттю:

1. Як елементна база всіх сучасних кластерів.

2. Як автономний засіб в якості ПЕОМ, серверів, робочих станцій.В останньому випадку сфера їх застосування різноманітна:

– 2D/3D САПР.

– Системи моделювання, засоби роботи з анімацією;

– Засоби обробки цифрових зображень.

– Електронні видавничі системи.

– Засоби відеомонтажу / рендеринга.

– Комп'ютерні ігри (на клієнтських комп'ютерах і серверах).

– Фінансове моделювання.

– Наукові та технічні розрахунки.

4.2.4. Процесори з індивідуальною пам'яттю

Для вирішення проблеми низької пропускної здатності пам'яті йдуть шляхом побудови на кристалі багатопроцесорної системи з індивідуальною пам'яттю для кожного ядра. Така система широко застосовується в обчислювальних кластерах. Це дає такі переваги:

При проектуванні використовувалися такі прогресивні технології:

– Використовувалися малі ядра. Такі ядра використовують, якщо їх кількість перевищує 100. Переваги малих ядер описані вище.

– Застосування об'ємного (тришарового) кристала. Зменшення розмірів кристала збільшує вихід придатних пластин.

– Для міжядерних обмінів використана матрична схема з'єднань, що використовує кремнієві лазери і оптичні лінії передачі даних, що забезпечують істотно більшу швидкість при менших енерговитратах, ніж електричні провідники. Для обміну використовуються протоколи інтернет.

– Одним з найбільш істотних переваг платформи є її "збірна конструкція". По суті Intel вдалося створити процесор, для якого не важливо який обчислювальний движок укладений в кожному з ядер. Це дозволить в майбутньому, використовуючи єдину логічну систему, створювати складні обчислювальні системи, які будуть збиратися як "конструктор", для вирішення самих різних завдань на базі однієї платформи.

Тришаровий кристал може виглядати наступним чином (малюнок взятий з (www.research.ibm.com / photonics):

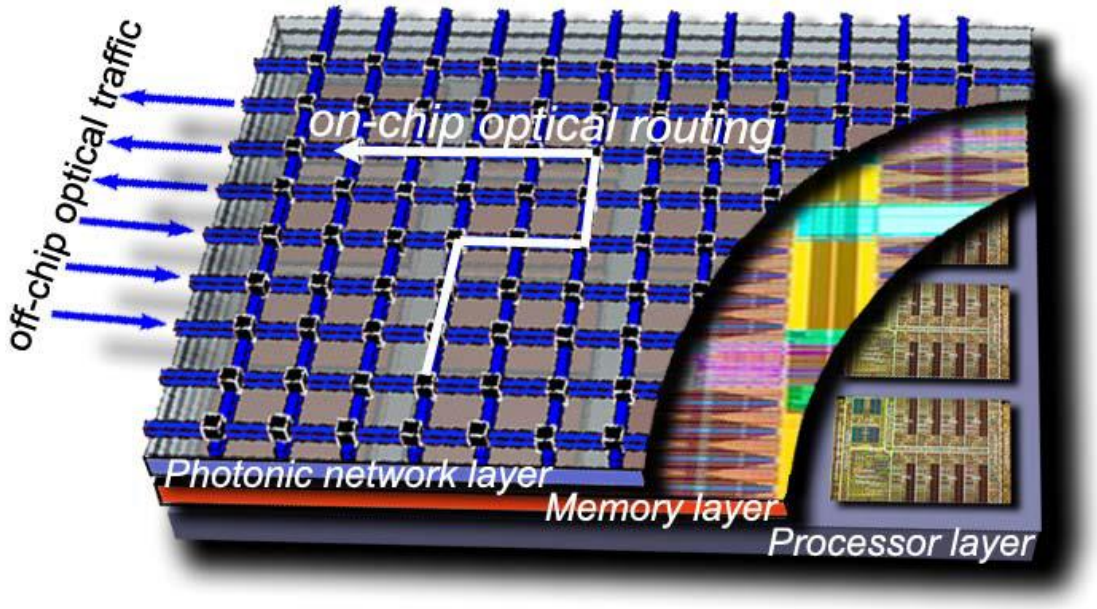


Рисунок 4.6. – Тришаровий кристал

Структура кристала являє собою своєрідний "сендвіч". Пам'ять розміщена в нижній частині чіпа і вертикально пов'язана з перебувають зверху ядром. Кожному ядру належить за 64 Мбайт ОЗУ. У прототипі загальний обсяг пам'яті склав 5 Гбайт, але Intel була обмежена технічними вимогами і певною кількістю транзисторів. Частково ця структура реалізована в процесорі Polaris.

4.2.5. Процесор Polaris на 80 ядер

Експериментальний мікропроцесор Polaris є результатом виконання компанією Intel програми Tera-Scale, в якій ставилися наступні завдання:

– TERA Operation Per Secord (1012 операцій в секунду).

– Terabit Data Transfer (Обмін даними при терабітових швидкостях).

– TERA I / O (Терабітовая швидкість операцій введення / виводу).

Polaris є кроком у напрямку створення мікропроцесорів для реалізації «хмарних» обчислень.

Загальні характеристики Polaris:

– 80 однопоточних ядер, будь ядро можна відключати при розвантаженні.

– Технології з типорозміром 65 нм.

– Продуктивність – 1.01 Tflops.

– 100 млн. транзисторів (Nehalem – 2.3 млрд).

– Розсіювана потужність 62 Вт

– Матриця процесорів 8 на 10 – 275, площа ядра – 3 кв.мм

– Частота – 3.16 ГГц

– Напруга ядер 0.95 В

– Матриця обмінів даними використовує мережеві протоколи, тому відмови окремих вузлів не вимикають весь кристал.

Мікроархітектура ядра. Ядро містить обчислювальний блок і і 5-портовий комутатор, що зв'язує його з чотирма сусідами.

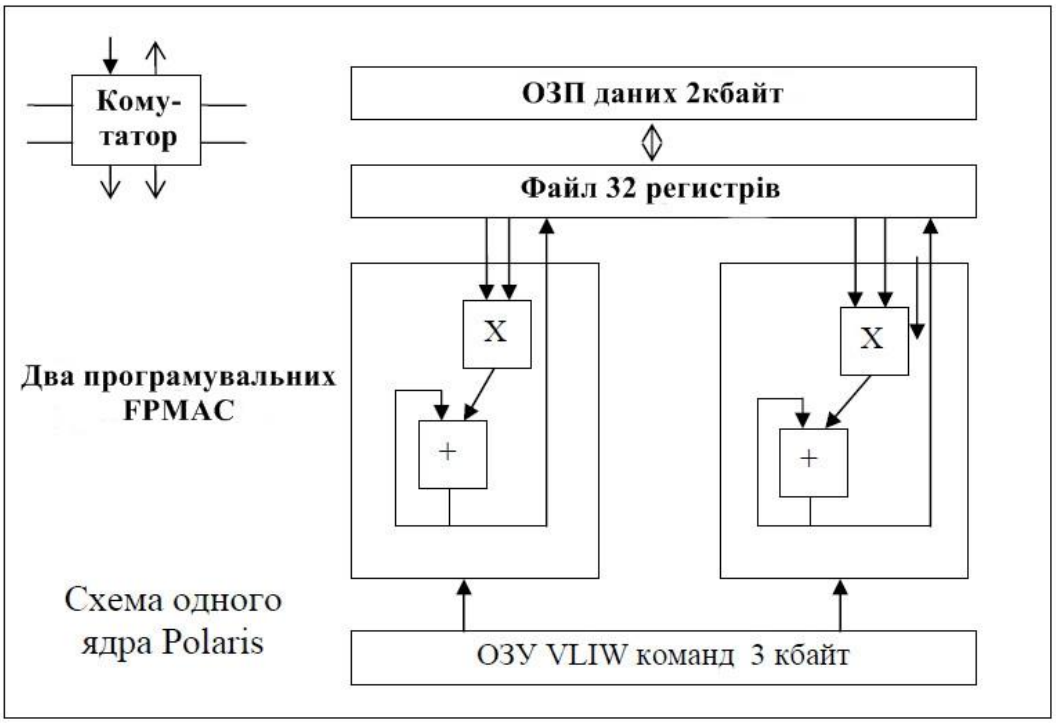


Рисунок 4.7. – Схема ядра Polaris

Ядро містить однотактні пам'ять даних (2 Кбайт) і команд (3 Кбайт), файл регістрів ємністю 32 рядки (десять портів читання плюс чотири порти записи) і два 32-розрядних FPMAC-пристрої з плаваючою комою. 9-стадійні конвеєри можуть виконувати команди «помножити-і-скласти», це дає чотири результату з плаваючою комою за такт на процесорний елемент, кожен з яких

використовує VLIW довжиною 96 біт, що забезпечує можливість виконання до восьми операцій за такт.

З метою максимального спрощення ядер вони побудовані на основі 96-розрядної архітектури VLIW (Very Long Instruction Word – "дуже довгого командного слова"). При цьому в одній команді VLIW може міститися до восьми простих операцій.

Архітектура кристала.

Пам'ять. Кожне ядро має окрему пам'ять. Для кожного вузла (4 ядра) є окрема пам'ять. Нарешті, не менш значущою інновацією є мікросхема статичної пам'яті L3 (SRAM) обсягом 20 Мбайт, пакетована з процесором і розміщена на одному кристалі з ним.

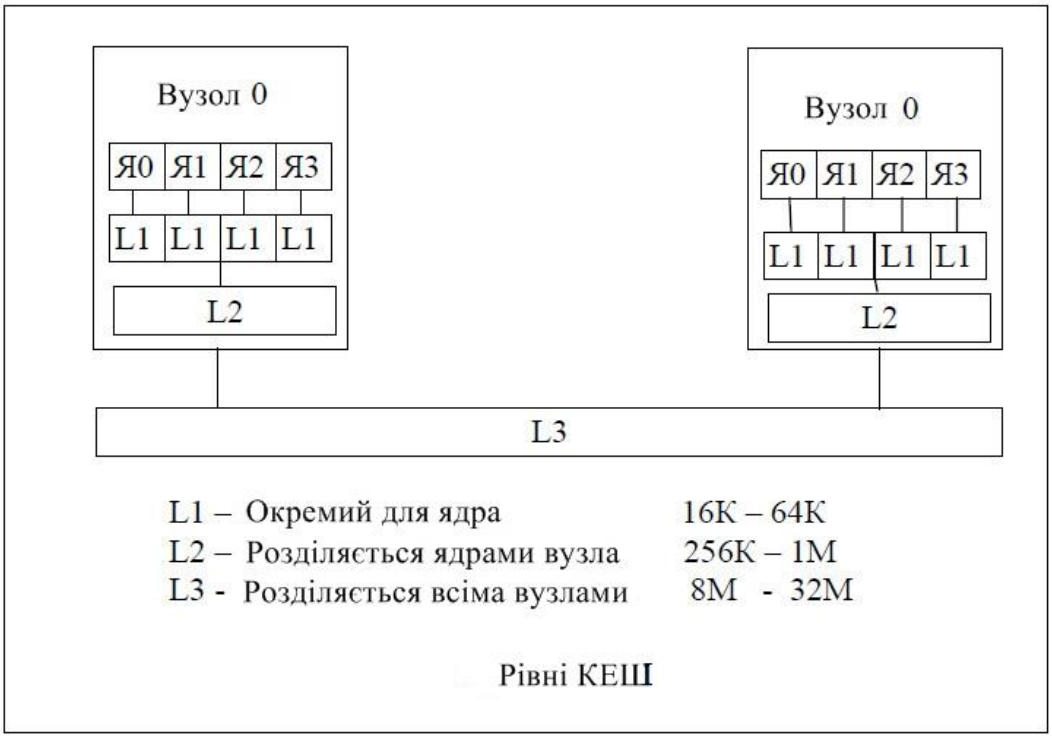


Рисунок 4.8. – Схема пам’яті ядра

Пакетування в процесорі дозволяє створити тисячі міжкомпонентних з'єднань і забезпечує смугу пропускання каналу між пам'яттю і ядрами шириною більше 1 Тбайт/с.

У процесорах на платформі Tera-scale передбачено використання трьох-рівневої системи кешування. Кеші L1 і L2 будуть безпосередньо пов'язані з кожним потоком і обсяг L1 складе від 16 до 64 Кбайт. Кеш другого рівня буде розміром 256 – 1024 Кбайт. Загальний для всього вузла стане кеш третього рівня об'ємом 8 – 32 Мбайт. Модулі кеш-пам'яті розміщуватимуться на тій же підкладці, що і обчислювальні ядра, маршрутизатори та інші елементи.

Також інженери Intel продемонстрували модель роботи нового L4-кеша високої ємності, доступного для всіх вузлів процесора, який розміщуватиметься між процесором і пам'яттю (схема "бутерброд"). Є також варіант розміщення кешу четвертого рівня навпроти процесора. У обох схем є свої переваги і свої недоліки. В даний час експерти компанії тестують обидва варіанти.

Комутація. Основною метою проекту Intel Terascale було дослідження можливості створення надкристальних мереж і управління енергоспоживанням. Кристал містить решітку 8x10 процесорних ядер. Канали між ядрами стійкі до фазовим зрушенням їх тактових сигналів.

Канал між двома ядрами підтримує функціонування двох логічних каналів (канал 0 і канал 1). Канал 0 зазвичай використовується для передачі коротких повідомлень, що містять команди, а канал 1 – для довгого повідомлення з даними. Це запобігає можливість затримки передачі команд.

Вбудований в кожне з ядер п’ятипортовий маршрутизатор використовується для обміну даними між ядрами, які, тим самим, об'єднуються в мережу. При цьому будь-яке ядро може використовуватися виключно для передачі даних, що дозволяє динамічно відключати живлення обчислювальних модулів таких ядер і економити електроенергію. Як видно на слайді, чотири з п'яти портів використовуються для зв'язку з іншими ядрами, а п'ятий – для підключення до вбудованої в "шаровий" чіп пам'яті. З метою оптимізації охолодження процесора сам багатоядерний чіп буде знаходитися зверху "бутерброд", а шари з пам'яттю – під ним.

Програмування.

Intel ® і інші виробники процесорів протягом декількох років невпинно пропагують поширення паралельних обчислень. Відповідно до закону Мура, через 10 років будуть проводитися процесори з 128 ядрами, через 12 років – з 256 ядрами, через 14 років – з 512 ядрами, а до 2023 року число ядер в процесорах може перевищити 1000.

На щастя, для створення якісно нових додатків для багатоядерних систем цілком згодяться перевірені часом і добре знайомі програмістам засоби розробки багатопроцесорного ПЗ: OpenMP, MPI і Pthreads.

Розробники Intel також трудяться над підготовкою нових моделей програмування. Детальну інформацію з розробки додатків для багатоядерних систем можна знайти на web-сайті мережі Intel ® Software Network.

Розробка ПЗ для терафлопних процесорів пов'язана і з іншими труднощами. Приріст продуктивності додатка в системі на базі терафлопного процесора визначатиметься часткою послідовного програмного коду (закон Амдала). Якщо програмний код на 25% складається з послідовних сегментів, то, теоретично, на необмеженому числі ядер такий додаток буде працювати в чотири рази швидше.

Закон працює і у зворотний бік: для досягнення максимальної продуктивності в чотирьохядерний системі досить розпаралелити код на 75 %. Щоб досягти максимального приросту продуктивності в системі з 64 і більше ядрами, необхідно розпаралелити код на 99 % і вище, що досить складно. Такого паралелізму не можна досягти, розпаралелив послідовний алгоритм автоматично за допомогою логічних Pthreads або OpenMP. Тут важливо переосмислити поставлене завдання і творчо підвести її рішення під новий, ефективний паралельний алгоритм.

4.2.6. Процесор SCC на 48 ядер

Перед експериментальним процесором Polaris ставилося завдання перевірити можливість побудови процесора з великою кількістю ядер. Ядра Polaris просто-напросто обчислювачі плаваючої точки, які не підтримують набір інструкцій IA і, відповідно, не можуть працювати як повноцінний центральний процесор.

Intel представила свою експериментальну розробку – 48-ядерний «одночіпний хмарний комп'ютер» (Single-chip Cloud Computer, SCC) з архітектурою IA-32. Якщо його разом зі спеціалізованою материнською платою встановити в комп'ютер і поставити на нього Windows, то система запрацює. У цьому і полягає його відмінність від процесор Polaris.

Мікрочіп SCC, виготовлений за 45-нм-технологічному процесу, містить 1,3 млрд транзисторів, частота межах 1,6-1,8 ГГц, а його енергоспоживання при максимальному навантаженні складає всього 125 Вт – стільки ж споживають дві звичайних лампи розжарювання.

SCC є спадкоємцем процесора Polaris, розробленого в рамках дослідницької програми Intel Tera-Scale, і попередником 100 – ядерного комерційного процесора. На відміну від свого прародителя SCC підтримує стандартне програмне забезпечення, створене для архітектурикс 86 – так, на демонстрації «хмарного» комп'ютера були успішно запущені операційні системи сімейств Windows і Linux. При цьому кожне з 48 ядер є незалежно програмованим, а кожне ядро може працювати на своїй напрузі і на своїй частоті або бути повністю вимкненим при відсутності навантаження.

«Хмарність» процесора полягає в тому, що всі 48 ядер сполучаються між собою як мережеві вузли, тому відмова деяких ядер не викликає відмови всього кристала.

Тепер демонструється вже не прототип 48-ядерного процесора, а реально працюючий комп'ютер на його основі. Втім, експериментальним комп'ютером справа явно не обмежиться. У Intel говорять, що багатоядерні концепції, які закладені в 48-ядерному процесорі, дозволяють використовувати похідні технології в пристроях від серверів до мобільних телефонів.

4.3. Підвищення продуктивності комп'ютерних систем за допомогою спеціалізованих процесорів

Використання в комп'ютерних системах апаратних прискорювачів не є новим підходом. Перші спеціалізовані засоби обчислювальної техніки з'явились в середині 60-х років, коли було набуто певного досвіду використання комп'ютерів у науковій та виробничій сферах. Тоді покращилась архітектура універсальних комп'ютерів, але їх швидкодія була недостатньою для розв'язання складних обчислювальних задач. Для таких потреб почали створювати спеціалізовані процесори, які апаратно реалізували алгоритми з найбільшою обчислювальною складністю і поєднувались до універсальних комп'ютерів. Ці перші спеціалізовані процесори були орієнтовані на виконання завдань цифрової обробки сигналів. Зокрема, такі спеціалізовані процесори

створювались для виконання алгоритмів згортки під час обробки даних сейсморозвідки. З часом спеціалізовані процесори стали входити до складу більшості серій універсальних комп'ютерів. Наприклад, до складу серій *IBM 360* і *IBM370* входили спеціалізовані процесори *AP2938* та *AP3838.* При цьому універсальний комп'ютер здійснював загальний контроль над обчислювальним процесом та операціями введення-виведення.

У цьому розділі наведено основні підходи до побудови спеціалізованих процесорів, перший з яких грунтується на використанні універсальних програмовних процесорів, спеціалізація яких реалізується на рівні програмного забезпечення, а другий – на використанні апаратно-орієнтованих процесорів. Показано переваги та слабкі місця цих підходів. Розглянуто питання прискорення універсальних комп'ютерних систем завдяки використанню в них спеціалізованих процесорів, які виконують обчислювальні алгоритми апаратним способом, тобто апаратно-орієнтованих. Наведено приклади структур таких спеціалізованих процесорів та сформовано вимоги до них у контексті їх використання у високопродуктивних обчислювальних системах на основі універсальних мікропроцесорів та реконфігуровних апаратних прискорювачів.

4.3.1. Підходи до побудови спеціалізованих процесорів

Сьогодні відомі два основні підходи до побудови спеціалізованих процесорів [10]. Перший грунтується на використанні універсальних програмовних процесорів, а особливості виконуваних задач враховують, по суті, за допомогою спеціалізації програмного забезпечення. Другий підхід ґрунтується на використанні процесорів, орієнтованих на виконувані алгоритми (функції) апаратним способом. Такі процесори називають функціонально- або апаратно-орієнтованими (в англомовній літературі – *ASIC – Application Specific Integrated Cirquit).*

Основною перевагою використання універсальних програмовних процесорів є гнучкість. Запис програм виконання заданого набору алгоритмів в пам'ять програм дає змогу створити спеціалізований процесор із заданими функціями. Такі процесори піддаються перепрограмуванню шляхом заміни вмісту пам'яті програм. Під час створення відповідних трансляторів програмне забезпечення таких спеціалізованих процесорів може бути написане мовами програмування високого рівня, що робить їх доступними для широкого кола користувачів. Крім того, значною перевагою цього підходу є можливість використання створених раніше програмних засобів. Разом з тим існують причини, через які використовувати універсальні програмовні процесори для побудови спеціалізованих процесорів може бути недоцільно.

По-перше – це висока трудомісткість розроблення, оскільки до складу спеціалізованого процесора, крім самого універсального процесора, необхідно ввести засоби для реалізації інтерфейсних функцій, синхронізації, розширення пам'яті програм і пам'яті даних тощо. Сам процес розроблення вимагає створення необхідного програмного забезпечення, наявності або створення технологічних програмних засобів для відпрацювання програм, а також

програмно-апаратних засобів для відлагодження апаратної та програмної частин спеціалізованих процесорів.

По-друге, універсальна архітектура може бути занадто надлишковою у функціональному і структурному аспектах для розв'язання однієї конкретної задачі. Це може призвести до надто високої споживаної потужності, збільшення ступеня інтеграції, кількості виводів корпусів та розмірів кристалів.

По-третє, універсальний процесор може не задовольняти вимог з продуктивності. Досягати необхідної продуктивності можна, будуючи багатопроцесорні спеціалізовані комп'ютерні системи, але їх використання може бути занадто дорогим. Тут йдеться не стільки про вартість багатопроцесорної системи, скільки про вартість допоміжних засобів, що забезпечують використання такої системи. Особливо багато проблем пов'язано з розробленням паралельних обчислювальних процесів, при реалізації яких всі процесори були б повністю завантажені. В такому разі системне програмне забезпечення повинне бути здатним оперативно розв'язувати задачу оптимізаційного планування завантаження процесорів багатопроцесорної системи конкретної архітектури. Такий підхід може вимагати великих витрат обладнання, що є небажаним.

Зараз, коли досягнення мікроелектронної технології підтритмуються потужними САПР, другий підхід, що передбачає створення апаратно-орієнтованих на виконувані алгоритми процесорів, є реальною альтернативою використанню універсальних програмованих процесорів.

По-перше, такий підхід забезпечує максимально можливу продуктивність під час розв'язання певної задачі.

По-друге, він вимагає мінімальних витрат обладнання на побудову спеціалізованого процесора для розв'язання задачі завдяки знаходженню компромісу між програмними та апаратними засобами. Таке спільне проектування апаратної і програмної частин отримало назву *hardware/software co-design* [11-13]. Однією із віток цього підходу є створення процесорів з спеціалізованою системою команд [1] (англійський термін *ASIP – Application-Specific Instruction Set Processor).*

По-третє, сучасні мови опису апаратних засобів, зокрема *VHDL* [14], а також засоби автоматизованого високорівневого синтезу мають настільки високий рівень, що процес проектування апаратно-орієнтованого процесора не є набагато складнішим ніж розроблення спеціального програмного забезпечення для універсального процесора. Якщо врахувати ще можливості сучасних САПР щодо забезпечення повного відпрацювання моделей електронних компонент, зокрема їх роботу в складі спеціалізованого процесора, а також використання бібліотек раніше створених електронних компонент, то стає очевидним, що настав час, коли другий підхід починає тіснити перший.

Отже, надалі розглядатимемо питання прискорення універсальних процесорів за допомогою використання спеціалізованих процесорів, орієнтованих на виконувані алгоритми (функції) апаратним способом, тобто апаратно-орієнтовані.

4.3.2. Архітектура апаратно-орієнтованих спеціалізованих процесорів

Зрозуміло, що архітектура апаратно-орієнтованих спеціалізованих процесорів залежить від особливостей алгоритмів, для виконання яких вони призначені. Історично першими були спеціалізовані процесори, призначені для розв'язання задач цифрової обробки сигналів. Тому зупинимося на розгляді їх архітектури. Аналіз архітектури спеціалізованих процесорів ЦОС свідчить, що їх арифметичною основою є операція обчислення виразу *a'Ь+c,* де *a* і *b* – вхідні операнди, *c* – результат попередньої операції. Прикладом може бути спеціалізований процесор *AP2938* [15], структурну схему якого наведено на рис. 4.9. До складу спеціалізованого процесора входять: арифметичний пристрій, оперативна пам'ять та пристрій управління. Арифметичний пристрій має три блоки: множення, додавання та нормалізації результату, які працюють у конвеєрному режимі одночасно і незалежно один від одного. Одночасно з арифметичним пристрєм працює пристрій управління, який здійснює зв'язок з універсальним комп'ютером та керує обчислювальним процесом, виконує введення-виведення та попередню обробку команд і даних, зокрема, перетворення чисел з формату з фіксованою комою на формат з рухомою комою і навпаки.

Вимоги розширення списку виконуваних операцій призвели до необхідності відокремлення помножувача і суматора. Таку структуру арифметичного пристрою мають, наприклад, спеціалізовані процесори *AP-190L, AP-120B, FPS-100* фірми *Floating Point Systems* [16]. Основні модулі та магістральну структуру цих процесорів показано на рис. 4.9.

Рисунок 4.9. – Структурна схема спеціалізованого процесора *AP2938*

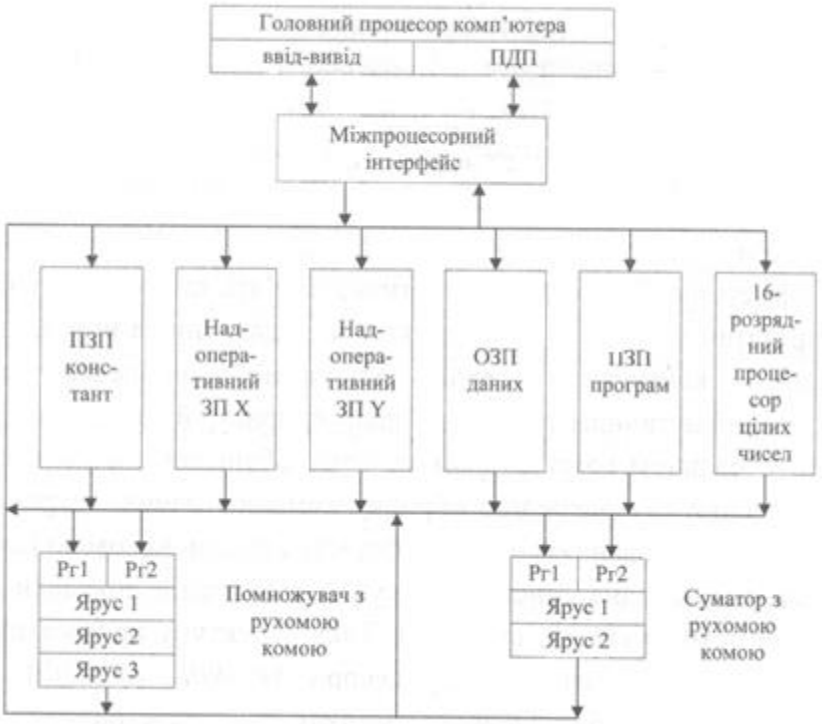


Рисунок 4.10. – Структура процесора *AP-120B*

У потоковому режимі конвеєрні помножувач і суматор, які містять відповідно три та два конвеєрні яруси, мають можливість видавати новий результат у кожному машинному циклі. Так, числа з рухомою комою множаться за три етапи, кожний з яких займає цілий машинний цикл. Формування добутку мантис двох операндів починається на першому етапі і закінчується на другому, а під час третього етапу додаються порядки обох операндів, а також нормалізується і заокруглюється добуток. У потоковому режимі внаслідок багаторазового суміщення сусідніх операцій множення результуюча швидкодія помножувача істотно підвищується. Як видно із рис. 4.11, хоча для виконання першої операції множення необхідно три машинні цикли, з початком потокового режиму новий результат виводиться в кожному машинному циклі.

Рисунок 4.11. – Конвеєрне виконання операції множення

У цих процесорах, крім розділених блоків пам'яті даних та програм, є два надоперативні запам'ятовувальні пристрої *X* та *Y* невеликої ємності, в яких зберігаються проміжні результати обчислень, а також ПЗП констант та процесор цілих чисел, який формує адреси даних у блоках пам'яті.

Необхідність підвищення швидкодії призвела до розпаралелювання каналів обробки даних і використання багатошинної організації передавання даних у наступних спеціалізованих процесорах ЦОС. В їхніх арифметичних пристроях паралельно працюють по декілька помножувачів і суматорів, наприклад, у спеціалізованому процесорі *AP3838* працює один чотириярусний помножувач і два чотириярусні суматори [17]. Така структура пристосованіша до алгоритму швидкого перетворення Фур'є (ШПФ) і алгоритмів комплексної арифметики. У спеціалізованому процесорі МАР-300 [18] паралельно працюють два арифметичні пристрої, в кожному з яких паралельно з'єднано помножувач і суматор, що дає змогу ще більше прискорити виконання алгоритмів ЦОС.

Високого рівня розпаралелювання досягнуто в спеціалізованому процесорі МАТР [19], який може мати в своєму складі від одного до чотирьох мікропрограм них процесорів, кожний з яких має власну пам'ять програм, а пам'ять даних є загальнодоступною (рис. 4.12). При цьому всі чотири процесори є конвеєрними.

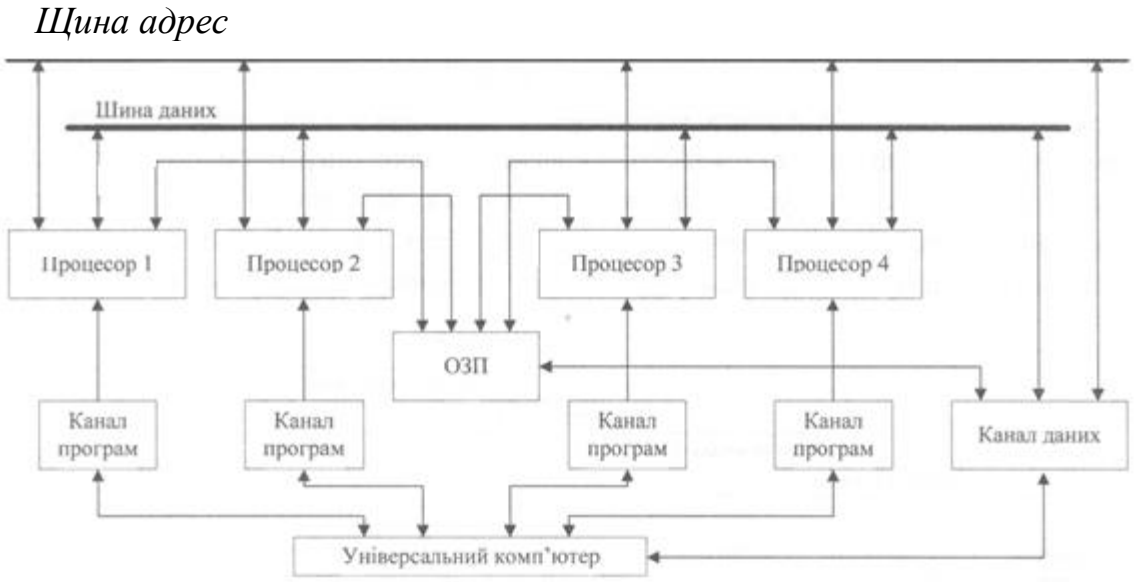


Рисунок 4.12. – Структура спеціалізованого процесора МАТР

З часом спеціалізовані процесори почали розробляти для прискорення виконання інших обчислювально складних алгоритмів. Наприклад, у Львівській політехніці було створено систему із спеціалізованих процесорів для прискорення виконання алгоритмів цифрової фільтрації, швидкого перетворення Фур'є та обчислення елементарних функцій. У процесорі обчислення елементарних функцій обробці підлягають числа, подані у форматі з фіксованою комою. Елементарні функції обчислюють за методом сегментної

апроксимації, відповідно до якого діапазон зміни аргументу [0,5; 1] поділяють на інтервали з подальшим наближенням функції на кожному інтервалі за допомогою виразу $Y = F\{X\} = A + W(X + B)^2$. Константи A та B вибирають за умови мінімізації абсолютної похибки, а константа W має дорівнювати степеню числа 2 (основа системи числення), що дає змогу замінити операцію множення операцією зсуву. На різних інтервалах константи мають різні значення. Кількість інтервалів також визначають за умови мінімізації абсолютної похибки, причому границі інтервалів визначаються к старшими двійковими розрядами аргументу, що полегшує здійснення адресації пам'яті, до якої записано константи. Операційну частину процесора, яка реалізує наведений вираз, показано на рис. 4.13.

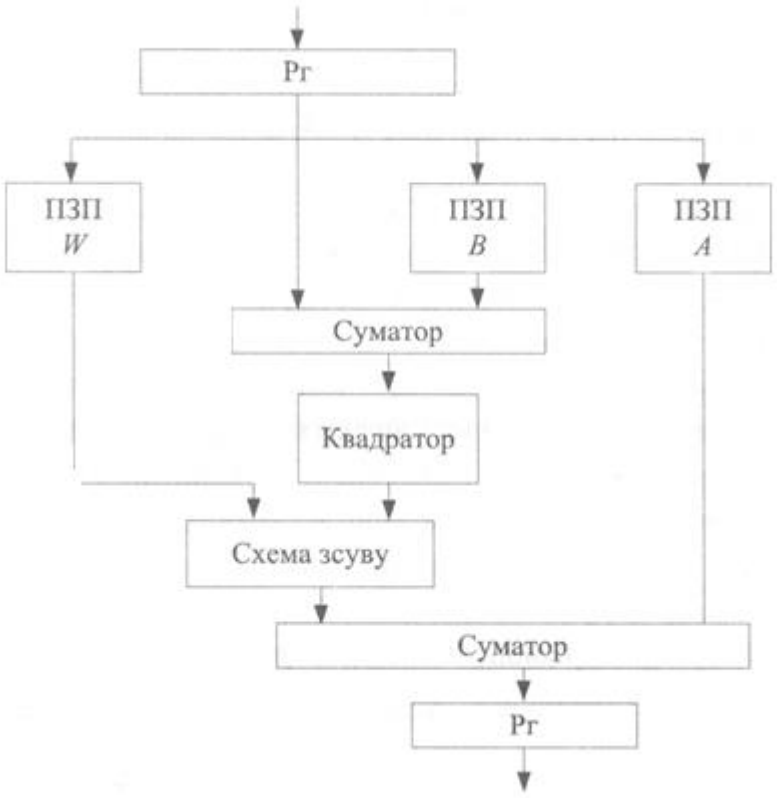


Рисунок 4.13. – Структура операційного пристрою процесора для обчислення функції

$Y = F(X) = A + W(X + B)^2$

Для реалізації операційного пристрою цього процесора необхідно три блоки постійної пам'яті для збереження коефіцієнтів, схему зсуву і два суматори. Для 12- розрядних операндів загальний обсяг ПЗП становить 8960 бітів. Константи A, B, W обчислюють з тим, щоб забезпечити задану точність обчислення. Вихідними даними для реалізації алгоритму розрахунку коефіцієнтів є: реалізована функція, величина інтервалу зміни аргументу, розрядність функції й аргументу. Після обчислення отримують такі величини: сумарна ємність ПЗП, число підінтервалів зміни аргументу, число розрядів, необхідних для кодування номера підінтервалу, значення констант A, B, W. Пристрій, що реалізує вираз $y = A + W(X + B)^2$, є багатофункціональним.

Для різних функцій константи мають різні значення. За допомогою конвеєра команд можна обчислювати на такому пристрої одночасно k функції від k даних.

Сьогодні сфери застосування комп'ютерів значно розширились, тому розширюються і функції спеціалізованих процесорів, а їхні характеристики покращились, насамперед завдяки успіхам мікроелектронної технології та розробленню нових архітектурних підходів.

4.3.3. Вимоги до спеціалізованих процесорів на основі кристалів програмовної логіки реконфігуровного прискорювача

Як було зазначено, сьогодні одним з підходів до побудови високопродуктивних обчислювальних систем є поєднання універсальних мікропроцесорів та реконфігуровних процесорних модулів, які апаратно реалізують заданий обчислювальний алгоритм і дають змогу змінювати конфігурацію завдяки тому, що реалізуються в кристалах програмовної логіки. Спеціалізовані процесори, які реалізуються в кристалах програмовної логіки реконфігуровного прискорювача, що взаємодіє з універсальним мікропроцесором, повинні відповідати ряду вимог. Вимоги до таких спеціалізованих процесорів можна поділити на такі групи [15, 20]: вимоги в частині його технічних характеристик; вимоги в частині організації приймання ним даних, їх опрацювання та видачі; вимоги в частині його архітектури та вимоги до кодів програмної моделі спеціалізованого процесора. Нижче ці вимоги розглядаються детально.

4.3.3.1. Вимоги до спеціалізованого процесора в частині його технічних характеристик

– Вимоги до спеціалізованого процесора в частині його продуктивності: він повинен володіти максимально можливою продуктивністю, якої можна досягти на виділеному для його реалізації обладнанні. Тобто має бути забезпечена максимальна ефективність використання обладнання реконфігуровного прискорювача.

– Вимоги до спеціалізованого процесора в частині затрат обладнання на його реалізацію: він повинен займати максимум з виділеного в програмовому кристалі реконфігуровного прискорювача місця під його реалізацію з тим, щоб найефективніше використати ресурси кристала. Наприклад, якщо під реалізацію виділено весь кристал, його проектують так, щоб його схема займала весь кристал.

– Вимоги до спеціалізованого процесора в частині споживаної ним потужності: він повинен споживати мінімальну потужність при виконанні заданих алгоритмів обробки даних із заданою продуктивністю.

Продуктивність спеціалізованого процесора відповідно до [21] розраховують так.

Нехай N елементів вхідних даних надходять у процесор K вхідними каналами з частотою F, і в процесорі має бути реалізований алгоритм з обчислювальною складністю $R = R(N)$, тобто R – це кількість операцій, які

потрібно виконати для реалізації алгоритму. Тоді необхідна для забезпечення обробки в реальному масштабі часу продуктивність процесора визначається з виразу $P = Kx\ F$ x RJN.

Для кожного алгоритму, який реалізується в процесорі, наперед відомими є кількість N вхідних даних та обчислювальна складність R алгоритму. Значення K та F задаються можливостями комп'ютера, до якого під'єднують процесор. Тому для кожного алгоритму можна наперед визначити продуктивність процесора при його реалізації з тим, щоб він забезпечував обробку даних з вхідних каналів у реальному масштабі часу.

Постає запитання: чи такої продуктивності можна досягти засобами процесора та як можна забезпечити вищеназвані вимоги? Для того, щоб розв'язати цю задачу, вважатимемо, що кількість обладнання реконфігуровного прискорювача дорівнює *QRA,* а кількість обладнання на реалізацію спеціалізованого процесора – *QSP.* Можливі чотири варіанти співвідношення між цими величинами. Якщо *QSP* незначно менше або дорівнює *QRA,* то мети досягнуто. Якщо ж *QSP* незначно більше за *QRA,* то потрібно дещо зменшити вимоги до продуктивності спеціалізованого процесора, що нескладно зробити, оскільки сьогодні існують засоби проектування програмних моделей спеціалізованих процесорів, які дають змогу задавати на етапі їх синтезу значення продуктивності або обмеження на затрати обладнання. Якщо виконується нерівність *QRA < QSP,* причому співвідношення між затратами обладнання більше за одиницю, то виникає потреба в скороченні затрат обладнання на спеціалізований процесор, чого досягають спрощенням його структури та, відповідно, пониженням продуктивності. При цьому для того, щоб комп'ютер не простоював, можна зменшити потоки інформації, скоротивши кількість вхідних каналів надходження даних на величину m = Integer *(QRA / QSP)* аж до одного, а також за потреби понизити частоту надходження даних до спеціалізованого процесора. Якщо ж виконується нерівність *QRA > QSP,* причому співвідношення між затратами обладнання більше за одиницю, то для ефективнішого використання ресурсів реконфігуровного прискорювача доцільно ввімкнути паралельно m = Integer(g5P / *QRA)* спеціалізованих процесорів.

Наведемо приклад. Для виконання в реальному масштабі часу алгоритму 1024-точкового швидкого перетворення Фур'є, для якого $R = 5NlogN,$ над даними, які надходять до реконфігуровного прискорювача lдвома каналами шини PCI з частотою 150 МГц кожний, процесор повинен мати продуктивність $\boldsymbol{P}$ = 7,5 x 10^9 операцій типу додавання/віднімання за секунду. Синтез процесора ШПФ такої складності зайняв 20 % від усього ресурсу кристала прискорювача. Отже, для повного використання ресурсів в кристалі можна задіяти 5 паралельно працюючих процесорів швидкого перетворення Фур'є.

4.3.3.2. Вимоги до спеціалізованого процесора в частині організації приймання даних, їх опрацювання та видавання

– Дані між центральним і спеціалізованим процесорами пересилають блоками певного розміру.

– Опрацьовувати дані можна під час пересилання (на фоні пересилання).

– Необхідно забезпечити таку швидкість пересилання даних, яка б давала змогу повністю використати ресурси процесора.

Зазвичай для виконання однієї команди здійснюють два пересилання даних: пересилання до спеціалізованого процесора блоку вхідних даних та пересилання до центрального процесора блоку вихідних даних.

4.3.3.3. Вимоги до спеціалізованого процесора в частині його архітектури

– Архітектура спеціалізованого процесора повинна враховувати особливості архітектури реконфігуровного прискорювача, на якому він реалізується.

– Доцільно реалізувати як програмні моделі процесорів апаратно-орієнтовані спеціалізовані процесори, оскільки за їх допомогою досягають гранично високої продуктивності за мінімальних значень затрат обладнання та споживаної потужності (як буде сказано в наступних розділах, технологія проектування спеціалізованих процесорів для їх реалізації в кристалах програмовної логіки передбачає розроблення програмної моделі процесора, наприклад, мовою опису апаратних засобів).

– В архітектурі спеціалізованого процесора необхідно враховувати можливості інтерфейсу між універсальним комп'ютером та реконфігу- ровним прискорювачем, на якому він реалізується.

4.3.3.4. Вимоги до кодів програмної моделі спеціалізованого процесора

Коди програмної моделі спеціалізованого процесора повинні:

– відповідати вимогам міжнародних стандартів;

– займати мінімальний об'єм пам'яті, в якій вони зберігаються;

– максимально швидко завантажуватись до реконфігуровного прискорювача.

4.4. Гібридні архітектури обчислювальних систем

Об'єми розрахунків, необхідних для розв'язання задач в різноманітних сферах науки та промисловості постійно, зростають, тому для їх розв'язання доводиться будувати все потужніші обчислювальні системи.

Причому стрімке зростання потужності окремих обчислювальних елементів (процесорних ядер) значно сповільнилось через досягнення пікових можливостей технологій їх виробництва. І до винайдення принципово нових технологій така тенденція залишиться незмінною.

Основним способом нарощування обчислювальної потужності сучасних суперкомп'ютерів стало об'єднання все більшої кількості обчислювальних елементів у велетенські обчислювальні комплекси.

Недоліком такого способу нарощування обчислювальних потужностей є те, що обчислювальні системи стають все більшими за розміром, а як наслідок більш складними, менш надійними та значно дорожчими в експлуатації, вже сьогодні для них змушені будувати окремі будівлі з розвиненою інфраструктурою, а споживання електроенергії стало співрозмірним з невеликими містечками.

Один із найпоширеніших методів оптимізації характеристик архітектури обчислювальної системи це її адаптація до області їх застосування. Звісно, універсальні обчислювальні системи є зручними через широкий спектр їх застосування, але такі високі показники ефективності, як при застосуванні спеціалізованих, для них є поки що недоступними.

Це пов'язано з тим, що кожен клас обчислювальних задач висуває специфічні вимоги до середовища їх виконання. Якщо в самій архітектурі передбачені додаткові можливості для виконання специфічних обчислювальних операцій, що часто використовуються при вирішенні задачі, то це дозволяє значно прискорити її виконання.

Найкращі результати розробок у цьому напрямку досягнуто при застосуванні гібридних архітектур, де в поєднанні з універсальними процесорами застосовуються спеціалізовані. Управління обчислювальним процесом здійснює універсальний процесор, а спеціалізовані використовуються як прискорювачі для виконання деяких специфічних операцій.

Пристрої для перетворення персональних комп'ютерів в маленькі суперкомп'ютери відомі досить давно. Ще в 80-х роках минулого століття на ринку пропонувалися так звані трансп'ютери, які вставлялися в поширені тоді слоти розширення ISA.

Спочатку їх продуктивність на відповідних задачах вражала, але потім зростання швидкодії універсальних процесорів прискорилося, вони посилили свої позиції в паралельних обчисленнях, і сенсу в застосуванні трансп'ютерів не залишилося. Хоча такі пристрої існують і нині – це різноманітні спеціалізовані прискорювачі. Але найчастіше сфера їх застосування вузька й поширення такі прискорювачі не отримали.

Але останнім часом естафета паралельних обчислень перейшла до масового користувача. Універсальні пристрої з багатоядерними процесорами

для паралельних векторних обчислень, що використовуються в 3D-графіці, досягають високої пікової продуктивності, недосяжної для універсальних процесорів.

Звісно, максимальна швидкість досягається лише в ряді зручних завдань і має деякі обмеження, але такі пристрої вже почали досить широко застосовувати сферах, для яких вони самого початку і призначалися. Відмінним прикладом такого паралельного процесора є процесор Cell, розроблений альянсом Sony-Toshiba-IBM, а також і всі сучасні відеоадаптери від лідерів ринку – компаній NVIDIA і AMD.

4.4.1. Огляд обчислень на графічних прискорювачах

Для 3D прискорювачів ще кілька років тому з'явилися перші технології неграфічних розрахунків загального призначення GPGPU (General-Purpose computation on GPUs). Сучасні ж графічні процесори містять сотні математичних виконавчих блоків, і ця потужність може використовуватися для значного прискорення безлічі прикладних програм зі складними обчисленнями.

Нинішні покоління GPU мають досить гнучку архітектуру, що разом з мовами програмування високого рівня і сучасними програмно-апаратними архітектурами розкриває їх додаткові можливості та робить доступнішими.

На створення GPGPU розробників спонукала поява досить швидких і гнучких шейдерних програм, що здатні виконувати сучасні графічні процесори. Розробники задумали зробити так, аби GPU розраховували не тільки зображення в 3D додатках, а й застосовувалися в інших паралельних розрахунках.

У GPGPU для цього використовувалися графічні API (Application Programming Interface): OpenGL і Direct3D, коли дані до графічного процесора передавалися у вигляді текстур, а розрахункові програми завантажувались у вигляді шейдерів. Недоліками такого методу є порівняно висока складність програмування, низька швидкість обміну даними між CPU і GPU.

Обчислення на GPU розвивалися і розвиваються дуже швидко. І далі, два основних виробника графічних процесорів, NVIDIA і AMD, розробили і анонсували відповідні платформи під назвою CUDA (Compute Unified Device Architecture) і CTM (Close To Metal чи AMD Stream Computing), відповідно.

На відміну від попередніх моделей програмування GPU, вони були виконані з урахуванням прямого доступу до апаратних можливостей відеоадаптерів. Платформи не сумісні між собою, CUDA – це розширення мови програмування C, а CTM – віртуальна машина, що виконує асемблерний код. Зате обидві платформи ліквідували деякі з важливих обмежень попередніх моделей GPGPU.

Звісно, відкриті стандарти, що використовують OpenGL, здаються найбільш переносимими і універсальними, вони дозволяють використовувати той самий код для графічних процесорів різних виробників, але в таких методів є маса недоліків, вони значно менш гнучкі й не такі зручні у використанні.

Крім того, вони не дають використовувати специфічні можливості певних відеоадаптерів, такі як швидка спільна пам'ять, що присутня в сучасних обчислювальних процесорах.

4.4.2. Різниця між CPU і GPU в паралельних обчисленнях

Зростання частот універсальних процесорів уперлося в фізичні обмеження і енергоспоживання, і збільшення їх продуктивності все частіше відбувається за рахунок розміщення кількох ядер в одному кристалі.

Пропоновані нині процесори зазвичай мають до восьми ядер (подальше зростання не буде швидким), вони призначені для звичайних додатків та використовують MIMD – множинний потік команд і даних. Кожне ядро працює окремо від інших, виконуючи різні інструкції для різних процесів.

Спеціалізовані векторні можливості (SSE2 і SSE3) для чотирикомпонентних (одинарна точність обчислень з плаваючою точкою) і двокомпонентних (подвійна точність) векторів з'явилися в універсальних процесорах передусім через збільшення вимог графічних додатків. Саме тому для певних завдань застосування GPU вигідніше, адже вони з самого початку зроблені для них.

Наприклад, у графічних процесорах NVIDIA основний блок – це мультипроцесор з 80-ма ядрами і сотнями ALU. В цілому з кількома тисячами регістрів і невеликою кількістю спільної пам'яті. Крім того, відеоадаптер містить швидку глобальну пам'ять з доступом до неї всіх мультипроцесорів, локальну пам'ять на кожному мультипроцесорі, а також спеціальну пам'ять для констант.

Найголовніше те, що ці кілька ядер мультипроцесора в GPU є SIMD (одиничний потік команд, множинний потік даних) ядрами. І ці ядра виконують одні й ті самі інструкції одночасно, такий стиль програмування є звичайним для графічних алгоритмів та багатьох наукових завдань, але вимагає специфічного програмування. Зате такий підхід дозволяє збільшити кількість виконавчих блоків за рахунок їх спрощення.

Отже, перелічимо основні відмінності між архітектурами CPU і GPU. Ядра CPU створено для виконання одного потоку послідовних інструкцій, а GPU проектують для швидкого виконання великого числа паралельно виконуваних потоків інструкцій. Універсальні процесори оптимізовано для досягнення високої продуктивності обробки єдиного потоку команд з цілими числами та числами з плаваючою точкою. При цьому доступ до пам'яті випадковий.

Розробники CPU намагаються досягти виконання якомога більшого числа інструкцій паралельно, для збільшення продуктивності. Для цього, починаючи з процесорів Intel Pentium, з'явилося суперскалярне виконання, що забезпечує виконання двох інструкцій за такт, а Pentium Pro відзначився позачерговим виконанням інструкцій. В паралельному виконанні послідовного потоку інструкцій є певні базові обмеження. Тому із збільшенням кількості виконавчих блоків кратного збільшення швидкості не досягти.

У графічних процесорів робота проста і розпаралелена з самого початку. Графічний процесор приймає на вході групу полігонів, проводить всі необхідні операції, і на виході видає пікселі. Обробка полігонів і пикселів незалежна, їх можна обробляти паралельно, не в комплексі.

Тому через початкову паралельну організацію роботи в GPU використовується велика кількість виконавчих блоків, які легко завантажити, на відміну від послідовного потоку інструкцій для CPU. Крім того, сучасні GPU також можуть виконувати більше однієї інструкції за такт. Так, архітектура Tesla у деяких умовах запускає на виконання операції MAD+MUL чи MAD+SFU одночасно.

GPU відрізняється від CPU ще за принципом доступу до пам'яті. У GPU він пов'язаний і легко передбачуваний – якщо з пам'яті читається піксель текстури, то через якийсь час прийде час і для сусідніх пікселів. Та й під час запису принцип той же – пиксель записується у фреймбуфер, і через кілька тактів буде записуватися той, що поруч із ним.

Тому організація пам'яті відрізняється від тієї, яку використовують в CPU. У графічному процесорі, на відміну від універсальних процесорів, просто не потрібна кеш-пам'ять великого розміру, а для текстур потрібні лише кілька (до 128-256 у нинішніх GPU) кілобайт.

Та й сама по собі робота з пам'яттю у GPU і CPU дещо відрізняється. Так, не всі центральні процесори мають вмонтовані контролери пам'яті, а у всіх GPU зазвичай є по кілька контролерів, аж до восьми 64-бітних каналів в чіпі NVIDIA GT200.

Крім того, на відеоадаптерах застосовується швидша пам'ять, і в результаті графічним процесорам доступна в кілька разів більша пропускна здатність пам'яті, що також дуже важливо для паралельних розрахунків, які оперують з величезними потоками даних.

В універсальних процесорах велика кількість транзисторів та площі кристалу займають буфери команд, апаратне передбачення розгалуження і величезні обсяги накристальної кеш-пам'яті. Усі ці апаратні блоки потрібні для прискорення виконання нечисленних потоків команд.

Графічні процесори витрачають транзистори на масиви виконавчих блоків, управляючі потоками блоки, спільну пам'ять невеликого обсягу і контролери пам'яті на кілька каналів. Перераховане вище не прискорює виконання окремих потоків, але воно дозволяє процесору обробляти одночасно кілька тисяч потоків, що виконуються одночасно та потребують високої пропускної здатності пам'яті.

Є багато відмінностей у підтримці багатопоточності. CPU виконує 1-2 потоки обчислень на одне процесорне ядро, а графічні процесори можуть підтримувати до 1024 потоків на кожен мультипроцесор, яких на кристалі декілька штук. І якщо переключення з одного потоку на інший для CPU варте сотні тактів, то GPU переключає кілька потоків за один такт.

Крім того, центральні процесори використовують SIMD (одна інструкція виконується над численними даними) блоки для векторних обчислень, а графічні застосовують SIMT (одна інструкція кілька потоків) для скалярної

обробки потоків. SIMT не вимагає, щоб розробник перетворював дані в вектори, і допускає довільні розгалуження у потоках.

Коротко можна сказати, що на відміну від сучасних універсальних CPU, відеоадаптери призначені для паралельних обчислень з великою кількістю арифметичних операцій. І значно більша кількість транзисторів GPU працює за прямим призначенням – обробкою масивів даних, а не управляє виконанням нечисленних послідовних обчислювальних потоків. На рис. 4.14 показано скільки місця на CPU та GPU займає різноманітна логіка.

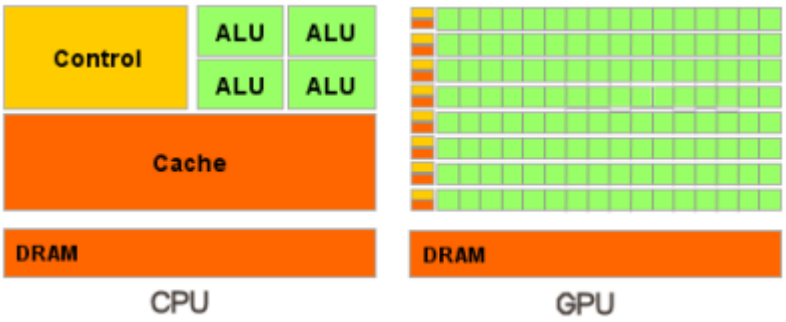


Рисунок 4.14. – Розмір логіки яка займає місце на CPU та GPU

У результаті, основою для ефективного використання потужності GPU в наукових або інших неграфічних обчисленнях є розпаралелювання алгоритмів на сотні виконавчих блоків, що наявні в відеоадаптерах. Наприклад, безліч додатків з молекулярного моделювання чудово пристосовані для розрахунків на графічних процесорах, вони вимагають великих обчислювальних потужностей і зручні для паралельних обчислень. А використання кількох GPU дає ще більше обчислювальних потужностей для вирішення таких завдань.

Виконання розрахунків на GPU показує відмінні результати в алгоритмах, що використовують паралельну обробку даних. Тобто, коли одну й ту ж послідовність математичних операцій застосовують до великого обсягу даних. При цьому кращі результати досягаються, якщо співвідношення числа арифметичних інструкцій до числа звернень в пам'ять досить велике. Це пред'являє менші вимоги до управління потоком, а висока щільність математики та великий обсяг даних відміняє необхідність у великих кешах, як на CPU.

У результаті всіх описаних вище відмінностей, теоретична продуктивність відеоадаптерів значно перевершує продуктивність CPU. Графік зростання продуктивності CPU і GPU за останні кілька років, створений компанією NVIDIA, наведено на рис. 4.15.

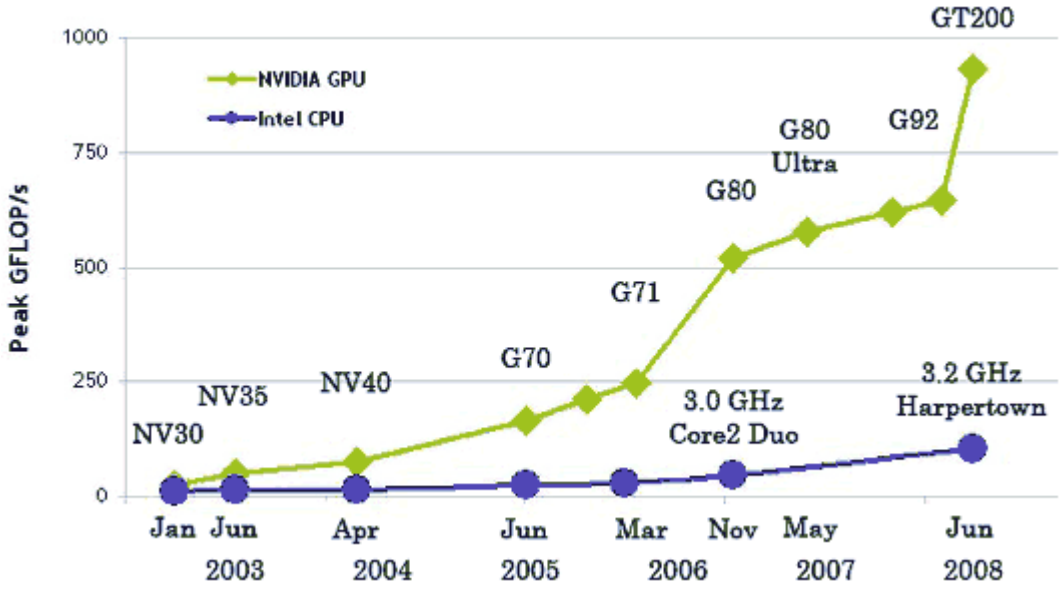


Рисунок 4.15. – Графік зростання продуктивності CPU і GPU за останні кілька років

Зрозуміло, ці дані не без частки лукавства. Адже на CPU набагато простіше на практиці досягти теоретичних цифр, та й цифри наведено для одинарної точності у GPU, і для подвійний – у CPU. У будь-якому випадку, для частини паралельних завдань одинарної точності вистачає, а різниця в швидкості між універсальними і графічними процесорами дуже велика.

4.5. Області застосування паралельних розрахунків на графічних процесорах

Щоб зрозуміти, які переваги приносить перенесення розрахунків на відеоадаптери, наведемо середні цифри, отримані дослідниками по всьому світу. У середньому, при перенесенні обчислень на GPU, у багатьох завдання досягається прискорення в 5-30 разів, в порівнянні зі швидкими універсальними процесорами.

Найбільші цифри (близько 100-кратного прискорення і навіть більше) досягаються на коді, який не дуже добре підходить для розрахунків за допомогою блоків SSE, але цілком зручний для GPU.

Приклади прискорень синтетичного коду на GPU проти SSE-векторизованого коду на CPU (за даними NVIDIA):

- Флуорисцентна мікроскопія: в 12 разів.
- Молекулярна динаміка: в 8-16 разів.
- Електростатика (пряма та багаторівнева сума Кулона): в 40-120 разів.

4.5.1. Технологія NVIDIA CUDA

Технологія CUDA (англ. Compute Unified Device Architecture) – це програмно-апаратна обчислювальна архітектура NVIDIA, заснована на розширенні мови Сі яка дає можливість доступу до набору інструкцій

графічного прискорювача і управління його пам'яттю при організації паралельних обчислень.

CUDA допомагає реалізовувати алгоритми, здійснимі на графічних процесорах відеоприскорювачів GeForce восьмого покоління і старше (серії GeForce 8, GeForce 9, GeForce 200), а також Quadro і Tesla.. В основі API лежить розширена мова Сі, а для трансляції коду з цієї мови до складу CUDA SDK входить компілятор командного рядка nvcc, створений на основі відкритого компілятора Open64.

Перерахуємо основні характеристики CUDA:

- уніфіковане програмно-апаратне рішення для паралельних обчислень на відеочипах NVIDIA;
- великий набір підтримуваних рішень, від мобільних до мультичиповых;
- стандартна мова програмування Сі;
- стандартні бібліотеки чисельного аналізу FFT (швидке перетворення Фурье) і BLAS (лінійна алгебра);
- оптимізований обмін даними між CPU і GPU;
- взаємодія з графічними API OpenGL і DirectX;
- підтримка 32- і 64-бітових операційних систем: Windows XP, Windows Vista, Linux і MacOS X;
- можливість розробки на низькому рівні.

CUDA включає два API: високого рівня (CUDA Runtime API) і низького (CUDA Driver API), хоча в одній програмі одночасне використовування обох неможливо, потрібно використовувати або один або інший. Високорівневий працює "зверху низькорівневого", всі виклики runtime транслюються в прості інструкції, що обробляються низькорівневим Driver API. Але навіть "високорівневий" API припускає знання про пристрій і роботу відеоадаптерів NVIDIA, дуже високого рівня абстракції там немає.

Є і ще один рівень, навіть більш високий – дві бібліотеки:

- CUBLAS – CUDA варіант BLAS (Basic Linear Algebra Subprograms), призначений для обчислень задач лінійної алгебри, що використовує прямий доступ до ресурсів GPU.
- CUFFT – CUDA варіант бібліотеки Fast Fourier Transform для розрахунку швидкого перетворення Фурье, що широко використовується при обробці сигналів. Підтримуються наступні типи перетворень: complex-complex (C2C), real-complex (R2C) і complex-real (C2R).

Розглянемо ці бібліотеки докладніше. CUBLAS – це перекладені на мову CUDA стандартні алгоритми лінійної алгебри, на даний момент підтримується тільки певний набір основних функцій CUBLAS. Бібліотеку дуже легко використовувати: потрібно створити матрицю і векторні об'єкти в пам'яті відеокарти, заповнити їх даними, викликати необхідні функції CUBLAS, і завантажити результати з відеопам'яті назад в системну.

CUBLAS містить спеціальні функції для створення і знищення об'єктів в пам'яті GPU, а також для читання і запису даних в цю пам'ять. Підтримувані функції BLAS: рівні 1, 2 і 3 для дійсних чисел, рівень 1 CGEMM для

комплексних. Рівень 1 – це векторно-векторні операції, рівень 2 – векторно-матричні операції, рівень 3 – матрично-матричні операції.

CUFFT – CUDA варіант функції швидкого перетворення Фурье що широко використовуються і є дуже важливою при аналізі – сигналів, фільтрації і т.п. CUFFT надає простий інтерфейс для ефективного обчислення FFT на відеочипах виробництва NVIDIA без необхідності в розробці власного варіанту FFT для GPU.

CUDA варіант FFT підтримує 1D, 2D, і 3D перетворення комплексних і дійсних даних, пакетне виконання для декількох 1D трансформацій в паралелі, розміри 2D і 3D трансформацій можуть бути в межах [2, 16384], для 1D підтримується розмір до 8 мільйонів елементів.

Середовище розробки CUDA (CUDA Toolkit) включає:

- компілятор nvcc;
- бібліотеки FFT і BLAS;
- профілювальник;
- відладчик gdb для GPU;
- CUDA runtime драйвер в комплекті стандартних драйверів NVIDIA;
- CUDA Developer SDK (початковий код, утиліти і документація).

Хоча трудомісткість програмування GPU за допомогою CUDA досить велика, вона нижче, в порівнянні з попередніми GPGPU рішеннями. Такі програми вимагають розбиття додатку між декількома мультипроцесорами подібно MPI програмуванню, але без розділення даних, які зберігаються в загальній відеопам'яті.

І оскільки CUDA програмування для кожного мультипроцесора подібне OpenMP програмуванню, воно вимагає хорошого розуміння організації пам'яті. Але, звичайно ж, складність розробки і перенесення на CUDA сильно залежить від додатку.

4.5.1.1. Переваги і обмеження CUDA

З погляду програміста, графічний конвейєр є набором стадій обробки. Блок геометрії генерує полігони, а блок растеризації – пікселі, що відображаються на моніторі. Традиційна модель програмування GPGPU виглядає наступним чином.

Щоб перенести обчислення на GPU в рамках такої моделі, потрібен спеціальний підхід. Навіть поелементне складання двох векторів зажадає відрисовки фігури на екрані або у екранний буфер. Фігура растеризується, колір кожного пікселя обчислюється за заданою програмою (піксельному шейдеру).

Програма прочитує вхідні дані з текстур для кожного пікселя, складає їх і записує у вихідний буфер. І всі ці численні операції потрібні для того, що в звичайній мові програмування записується одним оператором! Тому, застосування GPGPU для обчислень загального призначення має обмеження у вигляді дуже великої складності навчання розробників.

Та й інших обмежень достатньо, адже піксельний шейдер – це всього лише формула залежності підсумкового кольору пікселя від його координати, а

мова піксельних шейдеров – мова запису цих формул з Сі-подобним синтаксисом. Ранні методи GPGPU були хитрим трюком, що дозволяв використовувати потужність GPU, але без жодної зручності. Дані там представлені зображеннями (текстурами), а алгоритм – процесом растеризації. Потрібно особливо відзначити і вельми специфічну модель пам'яті і виконання.

Програмно-апаратна архітектура для обчислень на GPU компанії NVIDIA відрізняється від попередніх моделей GPGPU тим, що дозволяє писати програми для GPU на справжній мові Сі із стандартним синтаксисом, покажчиками і необхідністю в мінімумі розширень для доступу до обчислювальних ресурсів відеочипів. CUDA не залежить від графічних API, і володіє деякими особливостями, призначеними спеціально для обчислень загального призначення.

Переваги CUDA перед традиційним підходом до GPGPU обчислень:

- інтерфейс програмування додатків CUDA заснований на стандартній мові програмування Сі з розширеннями, що спрощує процес вивчення і упровадження архітектури CUDA;
- CUDA забезпечує доступ до спільної між потоками памяті розміром в 16 Кб на мультипроцесор, яка може бути використана для організації кеша с широкою полосою пропускання, в порівнянні з текстурними вибірками;
- більш ефективна передача даних між системною і відеопам'яттю;
- відсутність необхідності в графічних API з надмірністю і невигідними витратами;
- лінійна адресація пам'яті, і gather і scatter, можливість запису за довільними адресами;
- апаратна підтримка цілочисельних і бітових операцій.

Основні обмеження CUDA:

- відсутність підтримки рекурсії для виконуваних функцій;
- мінімальна ширина блоку в 32 потоки;
- закрита архітектура CUDA, належить NVIDIA.

Слабими місцями програмування за допомогою попередніх методів GPGPU є те, що ці методи не використовують блоки виконання вершинних шейдеров в попередній неуніфікованій архітектурі, дані зберігаються в текстурах.

4.5.1.2. Апаратні засоби з підтримкою NVIDIA CUDA

Усі відеокарти, що підтримують CUDA, можна використовувати для прискорення більшості вимогливих задач, починаючи від аудіо- і відеообробки, і закінчуючи медициною і науковими дослідженнями. Єдине реальне обмеження полягає в тому, що багато CUDA програм вимагають мінімум 256 мегабайт відеопам'яті, і це – одна з найважливіших технічних характеристик для CUDA-додатків.

На момент написання розрахунки CUDA підтримували всі продукти серій GeForce 200, GeForce 9 і GeForce 8, у тому числі мобільні продукти, починаючи з GeForce 8400M, а також і чіпсети GeForce 8100, GeForce 8200 і

GeForce 8300. Підтримку CUDA мають сучасні продукти Quadro і всі Tesla: S1070, C1060, C870, D870 і S870.

Особливо зазначимо, що разом з новими відеоадаптерами GeForce GTX 260 і GeForce GTX 280, були заанонсовані й відповідні рішення для високопродуктивних обчислень: Tesla C1060 і S1070 (представлені на рис. відповідно) У них використовується графічний процесор GT200, в C1060 він один, а в S1070 – чотири. На відміну від ігрових рішень, в них використовується по чотири гігабайти пам'яті на кожен чіп. З мінусів хіба менша частота пам'яті, ніж ігрових адаптерів, яка забезпечує по 102 гігабайт/с на чіп.

Рисунок 4.16. – NVIDIA Tesla C1060

Технічні характеристики NVIDIA Tesla C1060:

- Число GPU: 1.
- 240 потокових процесори.
- FPU-обчислення: IEEE 754 – з одинарною та подвійною точністю.
- Шина PCI Express x16 Generation 2.
- Інтерфейс пам'яті: 512-біт.
- Пропускна здатність пам'яті: 102 ГБ/с.
- Частота потокових процесорів: 1300 МГц.
- Частота пам'яті: 800 МГц.
- Об'єм GDDR3-пам'яті: 4096 МБ.
- Живлення: 2 x 6-контактних роз'єми або 6+8-контактні роз'єми.
- Енергоспоживання: 225 Вт /160 Вт (пікове/номінальне).
- Розміри: 26,46 x 11,03 см, двослотова плата.
- Продуктивність: 1 терафлопс.

Рисунок 4.17. – NVIDIA Tesla S1070

Технічні характеристики NVIDIA Tesla S1070:

- Число GPU: 4.
- 960 потокових процесори.
- FPU-обчислення: IEEE 754 – з одинарною та подвійною точністю.
- Системний інтерфейс: PCIe x16 або x8.
- Інтерфейс пам'яті: 4 x 512-біт.
- Пропускна здатність пам'яті: 408 ГБ/с.
- Частота потокових процесорів: 1500 МГц.
- Об'єм GDDR3-пам'яті: 16 ГБ.
- Енергоспоживання: 700 Вт.
- Формфактор: 1U.
- Продуктивність: 4 терафлопс.

Якщо за наданими виробником технічними характеристиками, розрахувати продуктивність на Ват споживання, то отримаємо близько 4,4 та 5,7 Гфлопс на Ват споживаної потужності відповідно для Tesla C1060 та Tesla S1070, що є чудовими показниками ефективності енерговикористання та значно перевищують подібні характеристики обчислювальних систем на універсальних процесорах.

4.6. Характеристики інтерконекту

Intel QuickPath Interconnect або просто QuickPath, скорочено QPI (раніше Common System Interface, CSI) – послідовна кеш-когерентна шина типу точка-точка для з'єднання процесорів між собою і з чіпсетом, розроблена фірмою Intel. QPI створювалась у відповідь на розроблену раніше консорціумом на чолі з фірмою AMD шину HyperTransport.

Шина QuickPath була створена для заміни застосовуваної раніше шини Front Side Bus, яка здійснювала зв'язок між центральним процесором і північним мостом материнської плати. Перші процесори з інтерфейсом QuickPath були випущені на ринок в 2008 році. Станом на початок 2010 року, зовнішній інтерфейс QuickPath використовується тільки в серіях процесорів Xeon і Core i7 з ядром Nehalem для роз'єму LGA 1366, а також буде використовуватися в наступному поколінні Itanium (ядро Tukwila). При цьому чипсети для роз'єму

LGA 1366 використовують шину DMI для зв'язку між північним і південним мостом. Процесори для роз'єму LGA 1156 не мають зовнішнього інтерфейсу QuickPath, оскільки чипсети для даного роз'єму підтримують тільки однопроцесорну конфігурацію, а функціональність північного мосту вбудована в сам процесор (і отже, для зв'язку процесора з аналогом південного мосту використовується шина DMI). Проте всередині процесора LGA 1156 зв'язок між ядрами і вбудованим контролером PCIe здійснюється через вбудовану шину QuickPath.

Кожне з'єднання шини QuickPath складається з пари односторонніх каналів, кожен з яких фізично реалізований як 20 диференціальних пар проводів. Дані передаються у вигляді пакетів (дейтаграм). Пропускна здатність одного каналу становить від 4,8 до 6,4 мільярда передач в секунду. Одна передача містить 16 біт корисного навантаження, отже теоретична сумарна пропускна здатність одного з'єднання (у двох напрямках) – від 19,2 до 25,6 гігабайт на секунду (тобто від 9,6 до 12,8 гігабайт / с в кожну сторону), при цьому один процесор може мати кілька з'єднань.

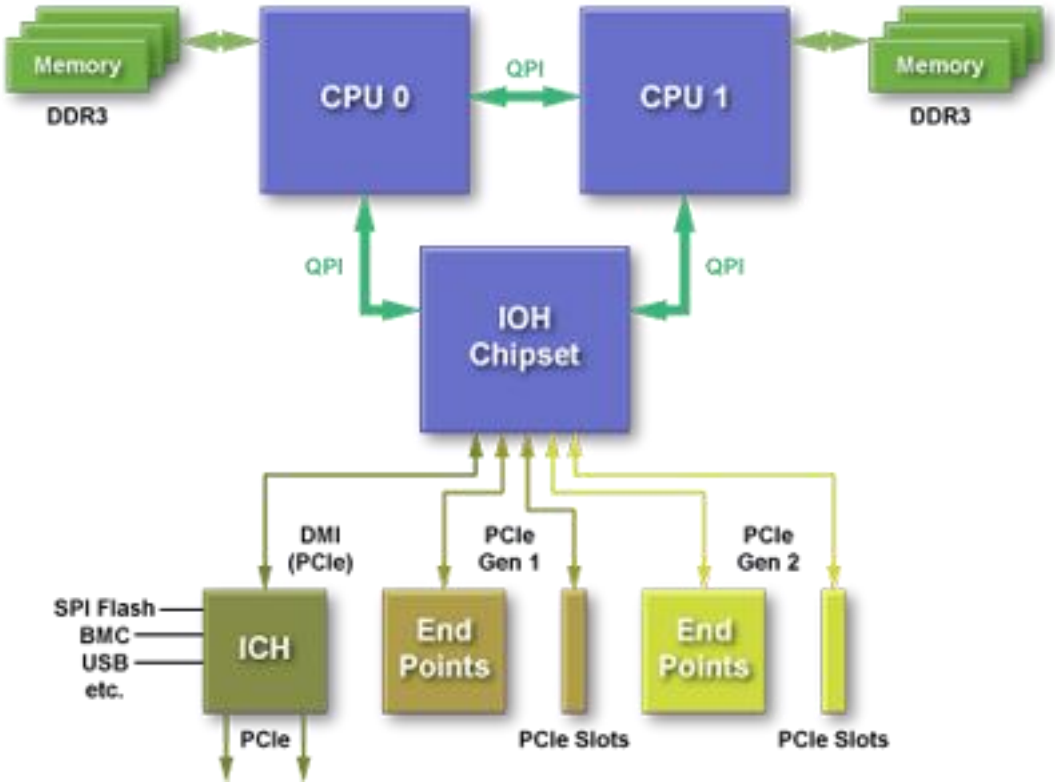


Рисунок 4.18. – Схема роботи інтерконекту

Служить для з'єднання пристроїв в системі між собою, а також для «спілкування» процесорів між собою в багатопроцесорних системах. Цьому сприяє її кеш-когеренстность (передача кеш-даних в обхід оперативної пам'яті на повній швидкості шини).

Схожі інтерфейси

Ідея подібних інтерфейсів не нова, і, наприклад, видання THG так описує походження цієї шини:

Рішення, вибране Intel під назвою QuickPath Interconnect (QPI), не є чимось новим; воно являє собою вбудований контроллер пам'яті і дуже швидку послідовну шину «точка-точка». Подібна технологія була представлена п'ять років тому в процесорах AMD, але насправді вона ще старше. Подібні принципи, які помітні в продуктах AMD і тепер Intel, являють собою результат роботи, виконаної десять років тому інженерами DEC під час розробки Alpha 21364 (EV7). Оскільки багато колишні інженери DEC перейшли в компанію з Санта-Клари, не дивно, що подібні принципи виплили в останній архітектурі Intel.

4.7. Обчислювальний кластер

Кластер. Магістральним напрямом розвитку паралельних ЕОМ для великоформатного паралелізму є побудова таких систем на базі засобів масового випуску: мікропроцесорів, каналів обміну даними, системного програмного забезпечення, мов програмування, конструктивів.

Системи з індивідуальною пам'яттю ідеально підходять підходять для реалізації на основі електронної та програмної продукції масового виробництва. При цьому не потрібно розробки ніякої апаратури. Такі системи отримали назву кластери робочих станцій або просто «кластери». Кластери також використовуються і в системах для розподілених обчислень Грід.

У загальному випадку, обчислювальний кластер – це набір комп'ютерів (обчислювальних вузлів), об'єднаних деякою комунікаційною мережею. кожен

обчислювальний вузол має свою оперативну пам'ять і працює під управлінням своєї операційної системи. Найбільш поширеним є використання однорідних кластерів, тобто таких, де всі вузли абсолютно одинакові за своєю архітектурою і продуктивності.

Кластер складається з окремих машин (вузлів) і об'єднуючої їх мережі (комутатора). Крім ОС, необхідно встановити та налаштувати мережеві драйвери, компілятори, програмне забезпечення для підтримки паралельного програмування і розподілу обчислювального навантаження.

Вузли кластера: підходящим вибором в даний момент є системи на базі процесорів Intel Core 2 Duo або Intel Core 2 Quad. Варто встановити на кожен вузол не менше 1Gb оперативної пам'яті. Бажано 2-4Gb. Одну з машин слід виділити в якості центральної (консоль кластеру) куди можна (але не обов'язково) встановити досить великий жорсткий диск, можливо більш потужний процесор і більше пам'яті, ніж на інші (робочі) вузли. Робити консоль кластеру більш потужною машиною має сенс, якщо необхідно мати на цьому комп'ютері крім інтерфейсу командного рядка більш зручне операційне оточення, наприклад віконний менеджер (KDE, Gnome), офісні програми, програми візуалізації даних і т.п.

Має сенс забезпечити захищений зв'язок цієї машини із зовнішнім світом. Іншими словами, мережа кластеру (мережа, яка складається з їх консолі кластеру та робочих вузлів) топологічно не повинна знаходиться всередині корпоративної мережі. Якщо необхідно забезпечити доступ до консолі кластеру

з корпоративної мережі або з мережі Інтернет, то в цьому випадку, зв'язок має йти через окрему мережеву карту, встановлену в головному комп'ютері, і окремий комутатор.

При комплектації робочих вузлів цілком можливо відмовитися від жорстких дисків – ці вузли будуть завантажувати ОС через мережу з центральної машини, що, окрім економії коштів, дозволяє налаштувати ОС і все необхідне ПЗ тільки один раз. Якщо ці вузли не будуть одночасно використовуватися в якості користувальницьких робочих місць, немає необхідності встановлювати на них відеокарти та монітори. Можлива установка вузлів в стійки (rackmounting), що дозволить зменшити місце, займане вузлами, але буде коштувати трохи дорожче.

Важливо зазначити, що бібліотеки для паралельних обчислень MPICH / MPI є кросплатформними, то вибір операційної системи (Windows vs Linux) не важливий. Однак слід врахувати той факт, що Linux є помітно менш ресурсномісткою системою. Наприклад при використанні PelicanHPC GNU Linux система займає в оперативній пам'яті не більше 40Мб! Вся інша пам'ять доступна паралельній програмі. Це дуже важливий чинник у тому випадку, коли кластер використовується з метою моделювання процесів на як можна більш докладній сітці.

Можлива організація кластерів на базі вже існуючих мереж робочих станцій, тобто робочі станції користувачів можуть використовуватися в якості вузлів кластеру вночі і в неробочі дні. Системи такого типу називають COW (Cluster of Workstations). У цьому випадку реальним видається варіант, коли кластер будується на основі існуючого комп'ютерного класу. Подібні класи вже є в більшості навчальних або наукових установах і зазвичай скомплектована однотипними машинами, що необхідні для кластеру. Проте зазвичай такі комп'ютерні класи працюють під операційною системою Windows і, ймовірно, для заміни її на Unix доведеться вирішити питання адміністративного плану і питання пов'язані з побудовою навчального процесу. Принципових перешкод для вирішення цих питань мабуть немає, оскільки Unix (конкретно Linux) має все необхідне програмне забезпечення для проведення навчального процесу чи наукової діяльності (компілятори, засоби розробки, офісні програми, програми роботи з зображеннями та візуалізації даних, засоби публікації).

Мережа: У найпростішому випадку для зв'язку між вузлами кластеру використовується один сегмент Ethernet (10Mbit/sec на витій парі). Проте дешевизна такої мережі, внаслідок колізій обертається великими накладними витратами на міжпроцесорний обміни, а хорошу продуктивність такого кластера можна чекати тільки на завданнях з дуже простою паралельною структурою і при дуже рідкісних взаємодіях між процесами (наприклад, перебір варіантів).

Для одержання гарної продуктивності міжпроцесорних обмінів використовують Fast Ethernet на 100Mbit/sec або Gigabit Ethernet. При цьому для зменшення числа колізій або встановлюють декілька "паралельних" сегментів Ethernet, або з'єднують вузли кластеру через комутатор (switch). Під паралельними сегментами мається на увазі така структура мережі, коли кожен

вузол кластера має більше однієї мережевої карти, які за допомогою спеціальних драйверів об'єднуються в один віртуальний мережевий інтерфейс, що має сумарну пропускну спроможність. Для того, щоб уникнути проблем з конфігуруванням такого віртуального інтерфесом, слід використовувати однакові мережеві карти на всіх машинах кластеру. Крім того, кожна паралельна лінія такого інтерфесу повинна являти собою Ethernet-мережу побудовану на окремому (від інших паралельних їй ліній) комутаторі.

Першим у світі кластером являється кластер Beowulf, створений в науково- космічному центрі NASA – Goddard Space Flight Center влітку 1994 року. Названий на честь героя скандинавської саги, кластер складався з 16 комп'ютерів. Особливістю такого кластера є масштабованість, тобто можливість збільшення кількості вузлів системи з пропорційним збільшенням продуктивності. Вузлами в кластері можуть служити будь серійно випускаються автономні комп'ютери, кількість яких може бути від 2 до 1024 і більше.

У СНД розвиток кластерів отримав в рамках програми Союзної держави Росії і Білорусії «СКІФ», розпочатої у 2000 році. В цей час побудовані кластери з швидкодією в десятки терафлопс.

Наступні роки вмістили в себе гігантський стрибок у розвитку апаратної бази, накопичення ідей і методів паралельного програмування, що зробило поява кластерів було неминучим. В даний час в списку Top 500 найбільш високопродуктивних систем саме кластери складають більшу частину списку.

У свою чергу кластери можна розділити на дві помітно відрізняються за продуктивності гілки:

– Кластери типу Beowulf, які будуються на базі звичайної локальної мережі ПЕОМ. Використовуються в вузах для навчальної роботи і невеликих організаціях для виконання проектів.

– Монолітні кластери, все обладнання яких компактно розміщено в спеціалізованих стійках масового виробництва. Це дуже швидкі машини, число процесорів в яких може досягати сотень і тисяч. Процесори в монолітних кластерах не можуть використовуватися в персональному режимі.

Однак, в обох випадках, кластер будується на продукції масового виробництва, зокрема, на локальних мережах. Основна відмінність кластерів від локальних мереж полягає в системному програмному забезпеченні (СПО). Таким пакетом зокрема є широко поширена бібліотеці функцій обміну для кластерів MPI (Message Passsing Interfase), описану вище.

Для реалізації цих функцій розробники MPI на основі можливостей ОС Linux створили спеціальний пакет MPICH, вирішальний цю та багато інших завдання. Такий же пакет зроблений і для ОС Windows NT.

Організація кластера. Приклад структури кластера на базі локальної мережі представлений на рис. 4.19 Кластерна система складається з:

– стандартних обчислювальних вузлів (процесори);
– високошвидкісної мережі передачі даних SCI;
– керуючої мережі Fast Ethernet / Gigabyte Ethernet;
– керуючої ПЕОМ.

– мережевої операційної системи LINUX або WINDOWS;

– спеціалізірованниз програмних средсв для підтримки обмінів даними (MPICH).

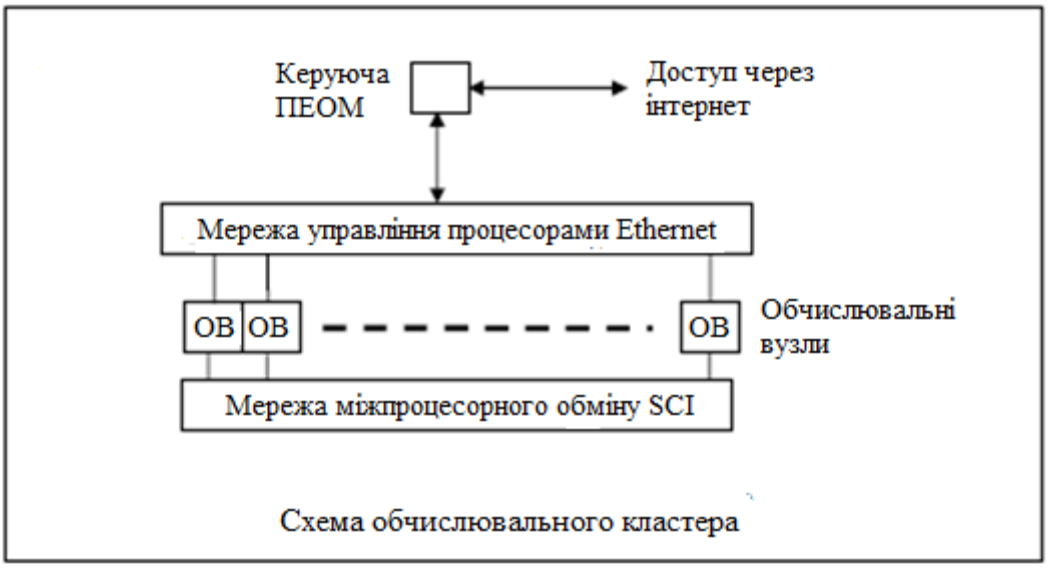


Рисунок.4.19. – Схема обчислювального кластера

Керуюча ПЕОМ кластера є адміністратором кластера, тобто визначає конфігурацію кластера, його секціонування, підключення і відключення вузлів, контроль працездатності, забезпечує прийом завдань навиконання обчислювальних робіт і контроль процесу їх виконання, планує виконання завдань на кластері. В якості планувальників зазвичай використовуються широко використовуючі пакети PBS, Condor, Maui та ін.

Для організації взаємодії обчислювальних вузлів суперкомп'ютера в його складі використовуються різні мережеві (апаратні і програмні) кошти, в сукупності утворюють дві системи передачі даних.

Мережа міжпроцесорного обміну об'єднує вузли кластерного рівня в кластер. Ця мережа підтримує масштабованість кластерного рівня Суперкомп’ютера, а також пересилання і когерентність даних у всіх обчислювальних вузлах кластерного рівня суперкомп'ютера у відповідності з програмою на мові MPI.

Мережа управління призначена для управління системою, підключення робочих місць користувачів, інтеграції суперкомп'ютера в локальну мережу підприємства і/або в глобальні мережі.

В якості обчислювальних вузлів зазвичай використовуються однопроцесорні комп'ютери, двох – або чотирипроцесорні SMP-сервери або їх багатоядерні реалізації.

Кожен вузол працює під керуванням своєї копії операційної системи, в якості якої найчастіше використовуються стандартні операційні системи: Linux, Windows та ін. Склад і потужність вузлів може змінюватися навіть в рамках одного кластера, даючи можливість створювати неоднорідні системи.

Найбільш поширеною бібліотекою паралельного програмування в моделі передачі повідомлень є MPI. Рекомендованої безкоштовної реацією MPI є пакет MPICH, розроблений в Аргоннській національній лабораторії.

MPI [8] є бібліотекою функцій межпроцессорного обміну повідомленнями і містить близько 300 функцій, які поділяються на такі класи: операції точка-точка, операції колективного обміну, топологічні операції, системні та допоміжні операції. Оскільки MPI є стандартизованою бібліотекою функцій, то написана із застосуванням MPI програма без переробок виконується на різних паралельних ЕОМ. MPI міститьбагато функцій, проте з принципової точки зору для написання переважної більшості програм достатньо кількох функцій, які наведені нижче.

4.7.1. Будова кластера

В загальному розгортання кластеру як такого – завдання актуальне та доволі не складне. Причому, для цього підійде будь-який дистрибутив. Який саме з дистрибутивів Linux ставити в якості базової ОС – не має значення. Ubuntu, Mandriva, Alt Linux, Red Hat, SuSE... Вибір залежить тільки від уподобань користувача.

Отже, щоб розвернути кластер, використовуючи дистрибутив загального призначення, слід виконувати наступне:

– Встановити операційну систему на комп'ютер, який буде виступати в ролі консолі кластеру. Тобто на цьому комп'ютері будуть компілюватися і запускатися паралельні програми. Іншими словами, за цим комп'ютером буде сидіти людина, запускати програми і дивитися, що вийшло.

– Після інсталяції базової ОС на консолі кластера, якщо це не зроблено в процесі первісної установки, необхідно буде встановити необхідні компілятори (фортран, С) і всі необхідні бібліотеки, desktop environment (GNOME або KDE), текстові редактори і т.д., тобто перетворити цей комп'ютер в робочу станцію розробника.

– Встановити з репозитарію або з вихідного пакет MPICH або OpenMPI.

– Описати в /etc/hosts майбутні вузли вашого кластеру, в тому числі і консоль кластеру.

– Встановити NFS і розшарити для всіх вузлів кластера якусь директорію, в якій будуть розміщуватися виконавчі модулі паралельних програм та файли даних, якими ці програми будуть користуватися в процесі своєї роботи.

– Встановити на консолі кластеру ssh-клієнт (обов'язково) та ssh-сервер (опціонально, якщо буде надаватися доступ до консолі кластеру по мережі).

– На всіх вузлах кластера встановити операційну систему, бібліотеки, необхідні для виконання користувальницьких паралельних програм, встановити MPICH, NFS-client, ssh-server. Вузли кластера в цілях економії ресурсів повинні навантажуватися в runlevel 3, так що ставити туди GNOME або KDE не треба. Максимум – поставити ряд бібліотек, якщо вони потрібні для користувача.

– Описати в /etc/hosts всіх вузлів кластера майбутні вузли вашого кластеру, в тому числі і консоль кластеру.

– На всіх вузлах кластера необхідно автоматично при завантаженні монтувати розшарений ресурс. Причому, шлях до цього ресурсу повинен бути однаковий, як на консолі кластера, так і на його вузлах. Наприклад, якщо на консолі кластеру дати доступ каталогу /home/mpiuser/data, то на вузлах кластеру цей ресурс також має бути змонтований в /home/mpiuser/data.

– На всіх вузлах кластера забезпечити безпарольний доступу по ssh для консолі кластеру.

Оскільки від мережі прямо залежить ефективність роботи кластеру, то хотілося б зробити наступне. Необхідно, щоб функціонування мережевої файлової системи NFS не заважало обміну даними, який здійснюють між собою частини паралельної програми, що працюють у різних вузлах. Щоб це здійснити, необхідно фізично розділити мережу на два сегменти. В одному сегменті буде працювати NFS, в іншому – відбуватиметься обмін даними між частинами програми.

Таким чином і в консолі кластеру та в його вузлах необхідно мати два мережевих інтерфейси (дві мережеві карти), відповідно, потрібно два набори світчів, не пов'язаних один з одним, і два набори мережевих реквізитів для цих інтерфейсів. Тобто, NFS працює, наприклад, в мережі 192.168.1.0/24, а обмін даними відбувається в мережі 192.168.2.0/24. І відповідно, у файлах /etc/exports і /etc/fstab повинні будуть бути прописані адреси з першої мережі, а у файлах /etc/hosts і в файлі machines.LINUX, що описують кластер – адреси з другої.

Існує дві доцільності, при яких використання кластеру є актуальним:

– Наявна різницева сітка розміру R, обчислення на якій при використанні звичайного комп'ютера займають час T. Час T – критичний параметр. Нам хочеться істотно зменшити час обчислень, маючи R як константу.

– Наявна різницева сітка розміру R, обчислення на якій при використанні звичайного комп'ютера займають час T. Час T – не критично. Нас цікавить збільшення розміру сітки понад наявною в одному комп'ютері пам'яті для більш детального рахунку, можливого отримання більш тонких ефектів і т.п.

Всі обчислення на різницевій сітці мають один спільний і важливий для нас параметр: час однієї ітерації. У разі використання кластеру цей час складається з двох частин: час рахунку на сітці Titer і час обміну даними між вузлами Texch. Titer залежить тільки від потужності процесора. А ось Texch залежить вже від розміру різницевої сітки, кількості вузлів кластеру та пропускної здатності мережі. Наведу остаточний результат для випадку двухгігабітної мережі, розміру різницевої сітки 64 гігабіти і часі однієї ітерації 100 сек.

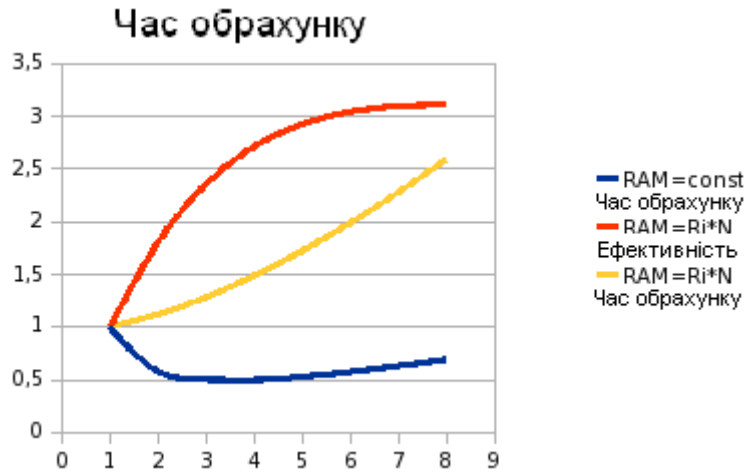


Рисунок 4.20. – Часова характеристика обрахунку даних

На графіку вісь ординат – тимчасова характеристика, вісь абсцис – кількість вузлів кластеру.

Необхідно звернути увагу на синю лінію. Це модель першого випадку, коли розбиваємо різницеву сітку постійного розміру на декілька вузлів. Як видно з графіка, час рахунку спочатку зменшується, при збільшенні кількості вузлів кластеру. Що ми й хотіли отримати. Однак зменшення відбувається до певної межі. При кількості вузлів більше чотирьох загальний час рахунку знову починає рости. Відбувається це через збільшення обсягу даних, що пересилаються між вузлами. Таким чином виходить, що при постійному розмірі сітки, збільшувати розмір кластеру понад чотири вузлів не має сенсу.

Тепер розглядається випадок 2, коли нам важливий розмір сітки, а з часом рахунку ми можемо знехтувати.

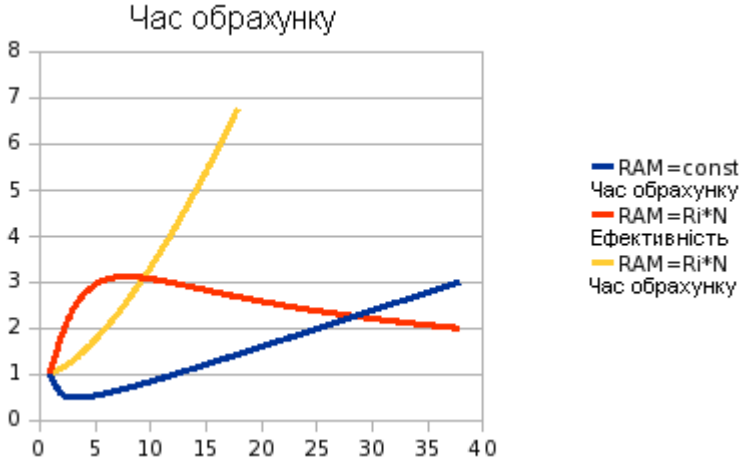


Рисунок 4.21. – Часова характеристика обрахунку даних

Давайте уявимо, що у нас є один комп 'ютер з необмеженою пам 'яттю. Збільшуючи розмір різницевої сітки, ми отримуємо лінійне збільшення часу рахунку. Тепер порівняємо цей час з тим, що вийде, якщо ми будемо вважати таку ж сітку, але на кластері. Причому збільшуючи розмір сітки вдвічі, ми збільшуємо удвічі і кількість вузлів кластера. Оскільки дві частини сітки

обраховуються паралельно, то час ітерації не збільшується, але з'являється час обміну даними. Червоний графік показує відношення часу рахунку на одному комп'ютері (з необмеженою пам'яттю) до часу рахунку такої ж сітки на кластері. Жовтий графік показує зростання часу обрахунку при збільшенні вузлів кластера (і, відповідно, збільшення розміру сітки). І зростання це, що важливо, менше, ніж лінійне.

Ми бачимо, що час рахунку на кластері істотно менший, ніж якщо б ми використовували сітку на одному комп'ютері. Причому, навіть при збільшенні розміру сітки (і вузлів кластера) в 40 раз, отримуємо виграш у часі.

Для кластерних обчислювальних систем одним із широко застосовуваних способів побудови комунікаційного середовища є використання концентраторів (hub) або комутаторів (switch) для об'єднання процесорних вузлів кластера в єдину обчислювальну мережу. У цих випадках топологія мережі кластеру представляє собою повний граф, в якому є певні обмеження на одночасність виконання комунікаційних операцій. Так, при використанні концентраторів передача даних в кожний поточний момент часу може виконуватися тільки між двома процесорними вузлами; комутатори можуть забезпечувати взаємодію декількох непересічних пар процесорів.

4.7.2. Організація мережі обчислювального кластеру

Мережа – це модульна і адаптуюча комутаційна система, яку можна налаштувати відповідно до самих різних вимог. Її модульність полегшує додавання нових компонентів або переміщення існуючих, а адаптивність спрощує внесення змін і удосконалень.

Мережа кластера нічим принципово не відрізняється від мережі робочих станцій, тому в найпростішому випадку для побудови кластеру необхідні звичайні мережеві карти та хаби / комутатори, які використовувалися б при облаштуванні якогось комп'ютерного класу. Однак, у випадку кластеру є одна особливість. Мережа кластера в першу чергу призначена не для зв'язку машин, а для зв'язку обчислювальних процесів. Тому чим вищою буде пропускна здатність мережі, тим швидше будуть вважатися паралельні завдання, запущені на кластері, отже робочі характеристики мережі набувають перчергове значення.

Для побудови обчислювальних кластерів використовують найрізноманітніше мережеве обладнання. При цьому, так як характеристики стандартних мережних пристроїв помітно поступаються характеристикам спеціалізованих комунікацій в "нормальних" MPP комп'ютерах, пропускна здатність мережі, що зв'язує вузли кластеру, у багатьох випадках виявляється вирішальною для продуктивності кластера. Використовуване мережеве обладнання характеризують зазвичай двома параметрами.

Латентність – це середній час між викликом функції передачі даних і самою передачею. Час витрачається на адресацію інформації, спрацювання проміжних мережних пристроїв, інші накладні витрати, що виникають при передачі даних.

Фактично пропускна здатність і латентність не тільки характеризують кластер, але й обмежують клас задач, які можуть ефективно вирішуватися на ньому. Так, якщо завдання вимагає частої передачі даних, кластер, який використовує мережеве обладнання з великою латентністю (наприклад GigabitEthernet), буде велику частину часу витрачати навіть не на передачу даних між процесами, а на встановлення зв'язку, в той час як вузли будуть простоювати, і ми не отримаємо значного збільшення продуктивності. Втім, якщо пересилаються великі обсяги даних, вплив періоду латентності на ефективність кластеру може знижуватися за рахунок того, що сама передача вимагатиме досить великого часу, може бути навіть у рази більшою за період латентності.

4.7.2.1. Мережеві карти

В якості мережевих адаптерів можна використовувати будь-які наявні в продажу карти, що підтримують роботу в стандартах 100BaseTx і GigabitEthernet. Що стосується списку переваг, то можна порекомендувати в першу чергу 3Com. Серед інших варіантів можна назвати Compex, Intel, Macronix, інші карти, підтримувані драйвером tulip, наприклад карти на чіпсетах DC21xxx. Особливо популярними при побудові кластерів являються плати на базі мікросхем Intel 21142/21143. Популярність цих карт викликана існуючим механізном високої продуктивності, в той час як їх ціна в порівнянні з конкуруючими пропозиціями зазвичай досить невелика. Що стосується мережевих карт фірми 3Com, то вони мають деякі переваги, помітно впливають на продуктивність мережевих комунікацій. Наведемо лише кілька прикладів можливостей апаратного забезпечення карт 3Com.

Розвантаження процесора при обчисленні контрольних сум TCP/UDP/IP. Звільняє центральний процесор від інтенсивних обчислень контрольних сум, виконуючи їх в самій мережевій платі. Тим самим підвищується продуктивність системи і час життя процесора.

Звільнення ЦП при відновленні сегментованих пакетів TCP: знижує навантаження на центральний процесор, підвищуючи продуктивність системи. Об'єднання переривань: дозволяє групувати декілька отриманих пакетів. Оптимізує обчислювальну ефективність хост-комп'ютера, скорочуючи число переривань і максимально звільняючи процесорні ресурси для роботи додатків.

Режим Bus mastering DMA: забезпечує більш ефективний обмін даними для зниження завантаження центрального процесора.

4.7.2.2. Комутатори

Другим важливим елементом мережі кластеру є пристрої комутації мережевих каналів. При виборі комутуючих пристроїв так само слід враховувати можливість використання channel bonding. Залежно від того, чи буде використовуватися технологія зв'язування каналів при побудові кластеру, можна зупинити свій вибір на різному мережевому обладнанні.

Комутатори та інші елементи мережевої структури використовуються для забезпечення комунікацій між процесорами, для підтримки паралельного

програмування і різних функцій управління. Для паралельного програмування організації взаємодії між процесами (Inter Process Communication, IPC) широко використовується комутатор Myrinet-2000 – дуже швидкий, добре масштабований широкосмуговий пристрій.

4.7.2.3. Мережеве забезпечення кластеру

Як вже говорилося, вузли кластеру можна зв'язати звичайним способом, використовуючи Ethernet-адаптери. З'єднання машин кластеру може виглядати так, як це показано на рисунку 1.3.

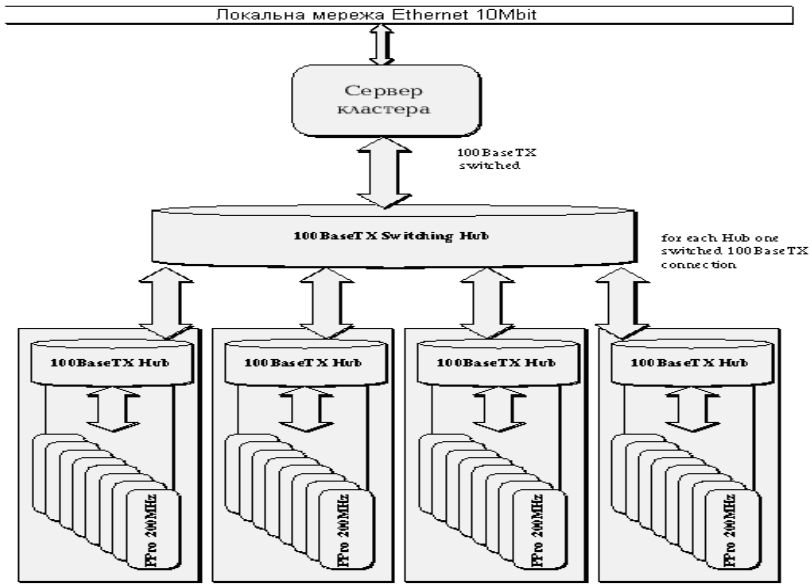


Рисунок 4.22. – З’єднання машин кластера

Проте швидкість доступу по мережі ніколи не буває занадто великою. Тому, для отримання від кластеру максимальної ефективності необхідно по можливості збільшувати пропускну здатність мережі. Для цього можна в машинах кластеру використовувати не одну, а кілька мережевих карт, об'єднавши їх в один логічний канал з більшою пропускною здатністю.

Інтерфейс користувача рівня для такого "злиття" каналів складається з двох програм: 'ifconfig' і 'ifenslave'. Перший мережевий інтерфейс конфігурується як зазвичай командою 'ifconfig'. Програма 'ifenslave' копіює установки першого інтерфейсу на всі інші додаткові інтерфейси. Цією ж командою можна при бажанні будь-які інтерфейси сконфігурувати в режимі Rx-only. Для і програм, що виконуються на кластері, метод абсолютно прозорий. Єдине вплив, який він надає – збільшення швидкодії.

Застосування методу має деякі обмеження: всі приєднані машини повинні мати однаковий набір bonded networks, тобто не можна в одній машині використовувати 2x100BaseTx, а в іншій 10Base і 100BaseTx. Застосування методу складається з двох частин, необхідні зміни Кернел для підтримки channel bonding, і програма ifenslave.

4.7.2.4. Мережева файлова система

Однією з особливостей запуску MPI-програм є необхідність наявності копій програми на всіх вузлах кластера, на яких вона виконується. Наприклад, якщо програма myprog розташована в каталозі /home/mpiuser/program1, то на всіх вузлах кластера повинен бути присутнім цей каталог і в нього повинна бути поміщена програма.

Ця умова вимагає необхідність яким-небудь чином розподілити копії виконуваного модуля програми між вузлами кластера. Аналогічна вимога стосується й зберігаються на диску даними, які програма буде використовувати.

Існують різні механізми, що дозволяють виконувати подібний розподіл. У більшості випадків це різноманітні скрипти, які здійснюють синхронізацію каталогів на вузлах кластера з допомогою команд scp або rsync. Подібні способи синхронізації мають свої недоліки. Наприклад, у випадку, коли різні копії програми повинні звертатися до одних і тих же даних, що зберігаються на диску, і змінювати їх певним чином, виникає проблема, пов'язана з необхідністю постійної синхронізації файлів на вузлах кластеру. Інша проблема виникає при використанні в якості вузлів кластеру бездискових станцій. У цьому випадку вся файлова система таких вузлів зберігається в оперативній пам'яті коп'ютера і чим більше ми завантажуємо даних на такий вузол, тим менше залишається пам'яті для виконання програми.

Для позбавлення від подібних проблем використовуються мережеві файлові системи. Існує велика кількість реалізацій таких систем, як платних, так і поширюваних під ліцензією GPL. Розглянема мережеву файлову систему NFS, наявну в будь-якому Linux-дистрибутиві загального призначення. Файлова система NFS – це аналог того, що й продукт Майкрософт відомий під назвою windows share.

Мережева файлова система NFS складається з двох компонентів: сервера і клієнта. Сервер здійснює мережевий доступ до каталогів базової файлової системи на основі певних правил розмежування доступу. Клієнт використовується для підключення до досутпних ресурсів.

4.7.2.5. Конфігурація сервера

Для забезпечення мережевого доступу вузлів кластеру до розшарених на сервері кластеру ресурсів необхідно спочатку вирішити підключення nfs-клієнтам до nfs-сервера. Далі будемо припускати, що вузли кластера мають ip-адреси в діапазоні 192.168.1.2-192.168.1.254. Консоль кластеру, до каталогів файлової системи до якої ми будемо підключатися через NFS, має ip-адреса 192.168.1.1. Для дозволу мережевого доступу до NFS з цих адрес у файлі /etc/hosts.allow прописуємо наступну рядок:

```
portmap: 192.168.1.1
```

Крапка в кінці рядка обов'язкова. Далі слід визначити до якого каталогу ми дозволяємо мережевий доступ. Тобто, якому каталогу давати доступ. Приміром, необхідно забезпечити у вузлах кластеру доступ до каталогу /home/mpiuser/data-and-progs. Для цього у файлі /etc/exports прописуємо рядок:

```
/Home/mpiuser/data-and-progs 192.168.1.0/255.255.255.0
(rw,no_root_squash)
```

На цьому налаштування серверної частини закінчено. Щоб зміни вступили в силу необхідно перезапустити службу NFS за допомогою команди "service portmap restart".

4.7.2.6. Конфігурація клієнтів

Переходимо від сервера кластера (консолі кластера) до решти вузлів. Все що буде описано нижче необхідно виконати на кожному комп'ютері кластеру крім консольного.

Для підключення будь-якої файлової системи (розділу диска, мережевого ресурсу) використовується команда mount, якщо підключення відбувається вручну, або запис у файлі /etc/fstab, якщо підключення відбувається в момент завантаження системи. Нас буде цікавити останній випадок.

Для забезпечення запуску mpi-програм нам потрібно, щоб вміст каталогу /home/mpiuser/data-and-progs на вузлах кластеру збігався з вмістом цього ж каталогу на консолі кластеру. Для цього необхідно в домашньому каталозі користувача mpiuser (/home/mpiuser) створити порожній каталог data-and-progs. Після чого прописати у файлі /etc/fstab такий рядок:

```
192.168.1.1:    /home/mpiuser/data-and-progs/home/mpiuser/data-and-
progs nfs rw 0 0
```

Щоб віддалений (мережевий) каталог монтувався автоматично при завантаженні кластеру, сервіс клієнта NFS повинен запускатися в процедурі початкового завантаження.

На цьому установка мережевої кластерної файлової системи завершена. При включенні кластеру, консоль кластеру повинна бути завантажена до того, як будуть включатися інші вузли.

4.7.2.7. SSH, беспарольный доступ

Розглянемо процедуру створення безпарольного доступу користувача user1 з консолі кластера на вузли кластеру по протоколу SSH. Безпарольний доступ забезпечить більш комфортну роботу в паралельній віртуальній машині. Так, відпаде необхідність вводити паролі доступу при додаванні в віртуальну машину кожного нового вузла і при копіюванні виконуваних модулів в локальні файлові системи вузлів кластеру.

Алгоритм забезпечення безпарольного доступу наступний:

1. Логін до консолі кластера: ssh user1@server.
2. Переходимо в каталог ssh: cd ~ /. Ssh.
3. Генеруємо rsa-ключі: ssh-keygen-t rsa.
4. На питання задати ім'я файлу тиснемо Enter – залишається ім'я за замовчуванням id_rsa.
5. На прохання поставити пароль тиснемо Enter два рази (другий раз для перевірки).
6. Копіюємо публічний ключ на вузол кластеру: scp id_rsa.pub user1@node1: ~ /.Ssh.

7. Логіном до вузла node1: ssh user1@node1.

8. Переходимо в каталог ssh: cd ~ /.Ssh.

9. Копіюємо публічний ключ: cat id_rsa.pub>> authorized_keys2.

10. Відключаємося від вузла node1.

11. Повторюємо пункти 6-10 для інших вузлів кластера (node2... nodeN).

Після проведення описаної процедури користувач "user1" зможе підключатися до вузлів кластеру з консолі кластера не вводячи свій пароль. Слід зазначити, таким чином забезпечується безпарольний доступ лише для одного користувача і лише в одному напрямку: консоль кластеру -> вузли кластеру. І тільки для випадку, коли user1 підключається до вузлів кластеру під своїм ім'ям.

4.7.3. Комунікаційні системи обчислювальних кластерів

Комунікаційні мережі [12]. Сполучна мережа, що реалізує одночасно (за один такт) всі види парних і колективних з'єднань, зображується у вигляді дводольного графа (рис. 5.2, а). Вона являє собою грати, на перетинах якої здійснюються необхідні замикання (рис. 5.2, б). Така решітка називається координатним перемикачем і має суттєвий недолік: об'єм обладнання в ній пропорційний N×M, тому координатні перемикачі використовуються для ПП розміром не більше 16... 32 процесора.

Комутатором, протилежним за своїми властивостями повного координатному з'єднувачу, є загальна шина UNIBUS, в якій осуществлятся тільки один обмін в одиницю часу. Між цими крайніми випадками є безліч мереж, що відрізняються призначенням, швидкодією і вартістю. Розглянемо деякі реальні системи обміну.

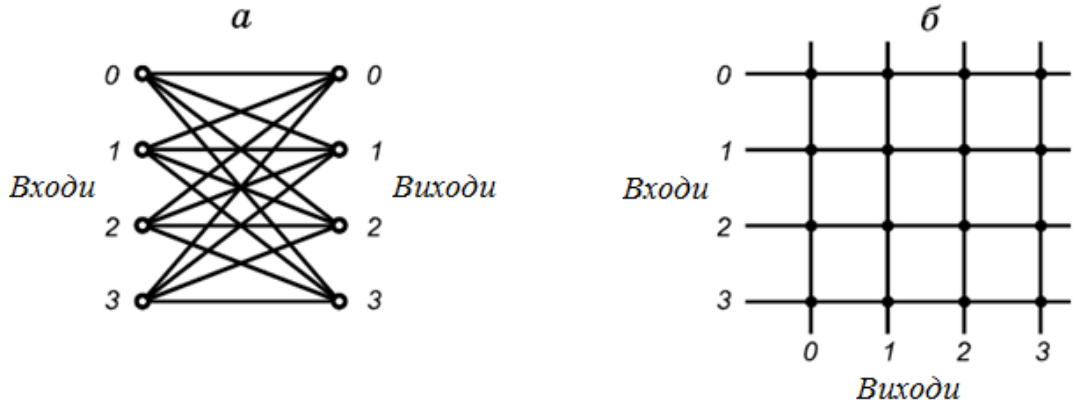


Рисунок 4.23. – Два подання координатного перемикача розміром 4×4:
а – у вигляді дводольного графа; б – у вигляді решітки зв'язків

Шинні технології – Ethernet. Найпростіша форма топології (bus) фізичної шини являє собою один основний кабель, оконцовані з обох сторін пеціальними типами роз'ємів – терминаторами. При створенні такої мережі основний кабель прокладають послідовно від одного мережного пристрою до іншого. Самі пристрої підключаються до основного кабелю з використанням

підвідних кабелів і Т-подібних роз'ємів. приклад такої топології наведено на рис. 4.24.

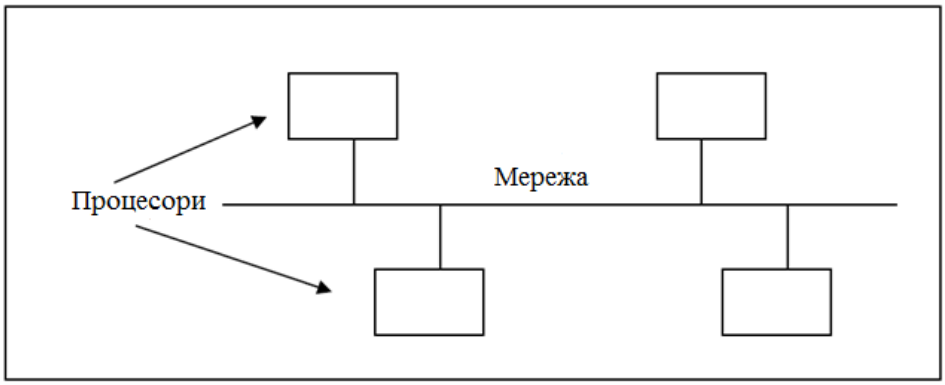


Рисунок. 4.24. – Комутатор із загальною шиною

Найбільш поширеною комунікаційної технологією для локальних мереж є технологія Ethernet, яка має кілька технічних реалізацій. Технологія Ethernet, Fast Ethernet, Gigabit Ethernet забезпечують швидкість передачі даних відповідно 10, 100 і 1000 Мбіт/с. Всі ці технології використовують один метод доступу – CSMA/CD (carrier-sense-multiply-access with collision detection), однакові формати кадрів, працюють в напів-і полнодуплексном режимах. Метод призначений для середовища, що розділяється всіма абонентами мережі.

На рис. 4.25 представлений метод доступу CSMA/CD. Щоб отримати можливість передавати кадр, абонент повинен переконатися, що середовище вільна. Це досягається прослуховуванням несучої частоти сигналу. Відсутність несучої частоти є ознакою вільності середовища.

На рис. 4.25 вузол 1 виявив, що середовище мережі вільна, і почав передавати свій кадр. У класичній мережі Ethernet на коаксіальному кабелі сигнали передавача вузла 1 поширюються в обидва боки, так що всі вузли мережі їх отримують. Усі станції, підключені до кабелю, можуть розпізнати факт передачі кадру, і та станція, яка дізнається свою адресу в заголовку кадру, що передається, записує його вміст у внутрішній буфер, обробляє отримані дані, передає їх нагору по своєму стеку, а потім посилає по кабелю кадр – відповідь. Адреса станції – джерела міститься у вихідному кадрі, тому станція – одержувач знає, кому послати відповідь.

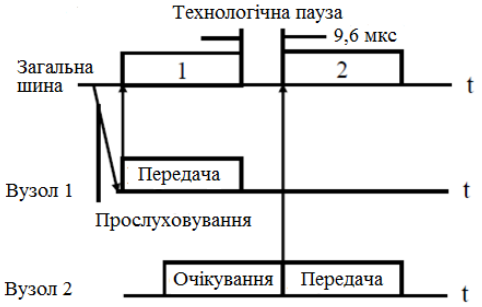


Рисунок 4.25. – Метод випадкового доступу CSMA/CD

Вузол 2 під час передачі кадру вузлом 1 також намагався почати передачу свого кадру, однак виявив, що середовище зайнята – на ній присутній

несуча частота, – тому вузол 2 змушений чекати, поки вузол 1 не припинить передачу кадру.

Після передачі кадру всі вузли зобов'язані витримати технологічну паузу в 9,6 мкс. Цей міжкадровий інтервал потрібен для приведення мережних адаптерів у початковий стан, а також для запобігання монопольного захоплення середовища однією станцією. Після закінчення технологічної паузи вузли мають право почати передачу свого кадру, оскільки середовище вільна. У наведеному прикладі вузол 2 дочекався передачі кадру вузлом 1, зробив паузу в 9,6 мкс і почав передачу свого кадру. У Ethernet передбачений спеціальний механізм для вирішення колізій.

Основний недолік мереж Ethernet на концентраторі полягає в тому, що в них в одиницю часу може передаватися тільки один пакет даних. Обстановка ускладнюється затримками доступу через колізій. Для виключення затримок такого роду використовуються проміжні комутатори – свічі(switch), тобто описані вище координатні перемикачі.

Кільцеві комутатори. Для кластерів боьшого розміру (кілька десятків або сотень вузлів) Ethernet виявляється повільним, тому використовують більш швидкі технології, наприклад, кільцеві. Найбільш відомим представником кільцевих структур є мережа SCI (Scalable Coherent Interface). Структура мережі представлена на рис. 4.26.

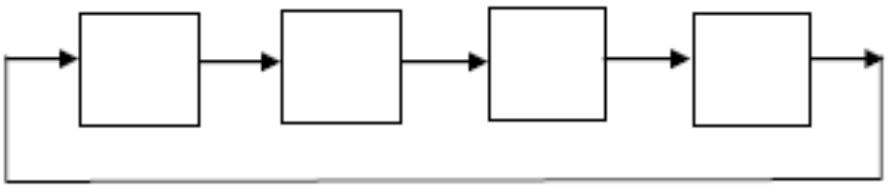


Рисунок 4.26. – Кільцева структура SCI

Кожен вузол має вхідний і вихідний канали. Вузли зв'язані односпрямованим каналами «точка-точка». При об'єднанні вузлів повинна обов'язково формуватися циклічна магістраль (кільце) з з'єднуються вузлів. Один вузол в кільці, званий «scrubber» (очисник), виконує функції знищення пакетів, що не знайшли адресата. Цей вузол позначає проходять через нього пакети і знищує вже помічені пакети. У кільці може бути тільки один scrubber.

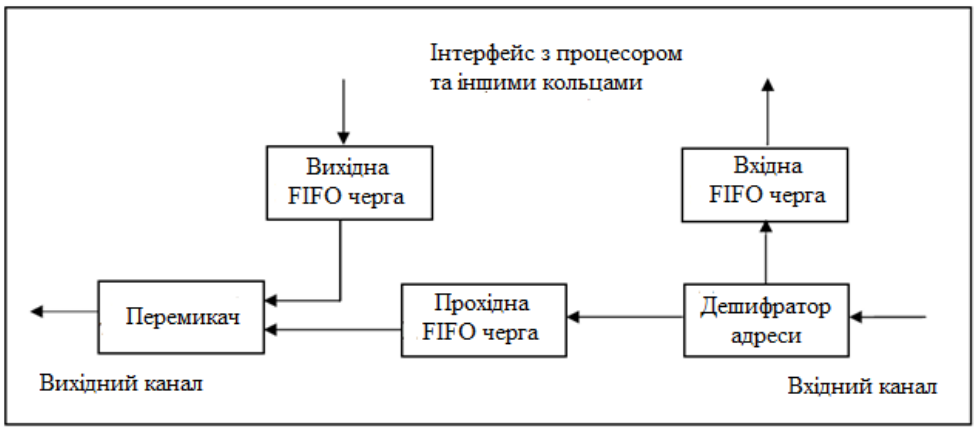


Рисунок 4.27. – Структура вузла SCI

Вузли SCI (рис. 4.27) повинні відсилати сформовані в них пакети з одночасним прийомом інших пакетів, адресованих вузлу, і пропуском через вузол транзитних пакетів.

Для транзитних пакетів, що прибувають під час передачі вузлом власного пакета, передбачена прохідна FIFO чергу. Розмір прохідний черзі повинен бути достатній для прийому пакетів без переповнення. У вузлі також вводяться вхідна і вихідна FIFO черзі для пакетів, прийнятих і переданих вузлом відповідно.

Вузол SCI приймає потік даних і передає інший потік данних. Ці потоки складаються з SCI пакетів і вільних (порожніх – iddle) символів. Через канали безперервно передаються або символи пакетів, або вільні символи, які заповнюють інтервали між пакетами. Вузол може передавати пакети, якщо його прохідна FIFO чергу порожня. Передача пакету ініціюється його переміщенням в вихідну FIFO чергу. Якщо протягом заданого часу не надходить відповідь пакет, то виконується по втор передачі пакета.

Вузол може послати багато пакетів (аж до 64), перш ніж буде отриманий у відповідь луна-пакет. Exo- пакети можуть приходити не в тому порядку, в якому були послані ініціювали їх пакети, тому необхідні номери пакетів для встановлення відповідності між пакетами і луна – пакетами.

Для запобігання блокувань всіх вузлів надається право доступу до SCI. Це означає, що всі можливі 64К пристроїв можуть почати передачу одночасно. Це і є масштабованість.

Головний недолік мережі SCI – фізичні обмеження на загальну довжину мережі. Тому SCI застосовується тільки для кластерів, оскільки вони розташовані на обмеженій території. SCI не підходить для Грід.

Таблиця 4.3.

Характеристики деяких технологій для обміну даними

Назва технології	Пікова пропускна здатність	Архтектура реалізації	Латентність на рівні MPI	Вартість
Etherrnet	12 MB/sec	Змішана	50 мкс	Низька
SCI	10 GB/sec	Кільце	4 мкс	Висока
Infiniband	120 GB/sec	Будь-яка	5 мкс	Середня

Час передачі повідомлення від вузла A до вузла B в кластерної системі визначається виразом T=S+L/R, де S – латентність, L – довжина повідомлення, а R – пропускна здатність каналу зв'язку. Латентність (затримка) – це проміжок часу між запуском операції обміну в програмі користувача і початком реальної передачі даних в комунікаційній мережі. Іншими словами – це час передачі пакета з нульовим обсягом даних. У таблиці 4.3 представлені характеристики деяких технологій для обміну даними.

ЛЕКЦІЯ 5. ПАРАЛЕЛЬНІ АЛГОРИТМИ, ЯК ЗАСІБ РОЗВ'ЯЗАННЯ ВЕЛИКИХ ЗАДАЧ НА ВИСОКОПРОДУКТИВНИХ СИСТЕМАХ. ГРАФ «ОПЕРАЦІЇ-ОПЕРАНДИ». ВИКОРИСТАННЯ БАГАТО ПОТОЧНОСТІ ПРИ ПРОГРАМУВАННІ ДЛЯ БАГАТОЯДЕРНИХ ПЛАТФОРМ

5.1 Паралельні алгоритми

Обчислювальні алгоритми володіють різним рівнем паралелізму. Для деяких класів задач є якісна оцінка величини паралелізму. Деякі результати такої оцінки представлені в наступній таблиці[13].

Характеристики деяких паралельних алгоритмів

Прихований паралелізм. Необхідна умова паралельного виконання i-ї і j-ї ітерацій циклу записується як і у випадку арифметичних виразу у вигляді правила Рассела [10] для циклів:

(OUT(i) AND IN(j)) OR (IN(i) AND OUT(j)) OR (OUT(i) AND OUT(j))=0

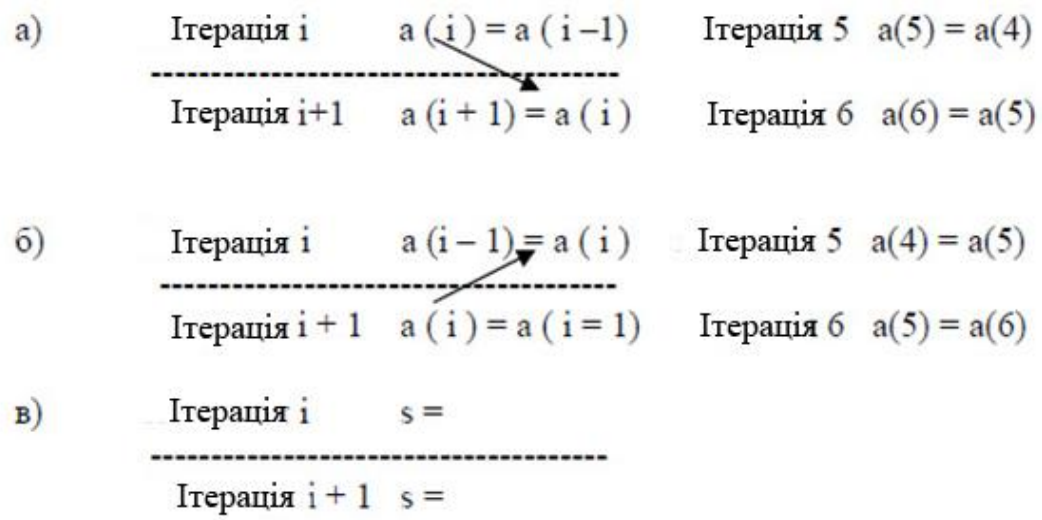


Рисунок 5.1. –Приклади залежностей між ітераціями – пряма (а), зворотна (б) і конкуреційна (в)

Таблиця 5.1.

Обчислювальні алгоритми

N пп	Найменування алгорита	Час обчислень	Число процесів
	Алгебра		
1	Рішення трикутної системи управління, звернення трикутної матриці	$0(\log^2 n)$	
2	Обчислення коефіцієнтів характеристичного рівняння матриці	$0(\log^2 n)$	$0(n^4 / \log^2 n)$
3	Розв*язання системи лінійних рівнянь перетворення матриці	$0(\log^2 n)$	$0(n^4 / \log^2 n)$
4	Метод виключення Гауса	$0(\log^2 n)$	$0(n^{\omega+1})$
5	Обчислення рангу матриці	$0(\log^2 n)$	поліноміальне
6	Схожість двох матриць	$0(\log^2 n)$	
7	Знаходження LU-розкладання симетричної матриці	$0(\log^3 n)$	$0(n^4 / \log^2 n)$
	Комбінаторика		
1	ε — оптимальний рюкзак, n - розмірність задачі	$0(\log n \log(n/\varepsilon))$	$0(n^3 / \varepsilon^2)$
2	Задача покриття з гарантованою оцінкою відхилення на більш, ніж $(1+\varepsilon)\log d$ разів	$0(\log^2 n \log m)$	$0(n)$
3	Знаходження ϵ - орошої розкраски в задачі про балансування множин	$0(\log^3 n)$	поліноміальне
	Теорія графів		
1	Ранжировка списку	$0(\log n)$	$0(n/\log n)$
2	Ейлерів шлях в дереві	$0(\log n)$	$0(n/\log n)$
3	Відтискання дерева з мінімальною вагою	$0(\log^2 n)$	
	Транзитивне замикання	$0(\log^2 n)$	
5	Розмалювання вершини у $\Delta+1$ і Δ кольорів	$0(\log^3 n \log\log n)$	$0(n+m)$
6	Дерево пошуку в глибину для графа	$0(\log^3 n)$	$0(n)$
	Сортування і пошук		
1	Сортування	$0(\log n)$	$0(n)$
2	Злиття для двох масивів позміром n і m $N = m+m$	$0(\frac{N}{P} + \log N)$	$P = 0(N / \log N)$

Знання оцінок з таблиці не дає ще можливості побудувати практичний паралельний алгоритм. У багатьох випадках він не очевидний і його ще потрібно різними прийомами проявити у формі, доступній для програмування на деякій паралельні мові. Приклад такого прояву наводиться нижче.

Розглянемо метод гіперплоскостей, запропонований L.Lamport в 1974 році на прикладі розв’язку рівнянь в приватних похідних. Метод зветься "Фронт хвилі ". Нехай дана програма для обчислення в циклі значення Xi, j, як середнього двох суміжних точок (зліва і зверху).

```
DO 1  I = 1,N
DO 1  J =  1,N
1   X (I, J) = X (I-1, J) + Y (I, J –1) + C
```

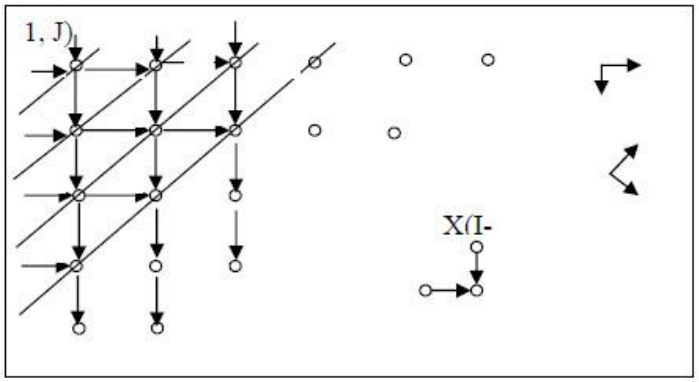


Рисунок 5.2. – Метод гіперплоскостей

Розглянемо дві будь-які суміжні за значеннями індексів ітерації, наприклад:

X (2,2) = X (1,2) + X (2,1)

X (2,3) = X (1,3) + X (2,2)

Рисунок показує, що між ітераціями існує пряма залежність. Використовувати для складання суміжні рядки не можна, так як нижній рядок залежить від верхнього. Не можна складати і суміжні стовпці, так як правий стовбець залежить від лівого. Проте паралелізм в задачі є, наприклад, всі операції в діагоналі 41, 42, 43, 44 можна виконувати паралельно.

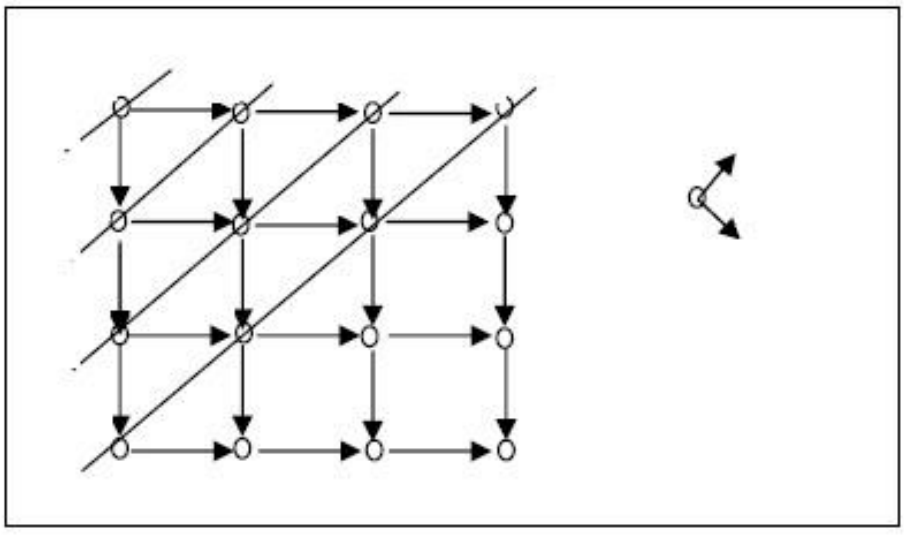


Рисунок 5.3. – Метод гіперплоскостей при повороті на 45 градусів

Якщо повернути осі i, j на 45 градусів і перейменувати операції всередині кожної діагоналі, як на наступному рисунку, то можна використовувати цей паралелізм. Відповідна програма для верхніх діагоналей (включаючи головну) і нижніх діагоналей наведена нижче.

```
  DO 1 I = 1,N
  DO PAR J = 1, I
  X (I, J) = X(I-1, J) + X(I-1, J-1)
1 CONTINUE
-------------------------------------
  K + 2
  DO 2 I = N +1, 2N – 1
  DO PAR J + K, N
  R = K + 1
  X(I, J) = x(I-1, J) + X(I-1, J-1) + C
2 CONTINUE
```

Нехай I =3, тоді x(3,1)=x(2,1)+x(3,0)
x(3,2)=x(2,2)+x(2,1)
x(3,3)=x(2,3)+x(2,2)

Ці ітерації справді незалежні

Тут К = 1,2 позначає лівий – верхній, або правий – нижній трикутник простору ітерацій.

Алгоритм методу гіперплоскостей полягає в наступному:

– Виробляється аналіз індексів і побудова залежностей в просторі ітерацій.

– Визначається кут нахилу осей і перейменування змінних.

– Будується паралельна програма.

Недолік методу гіперплоскостей полягає в тому, що ширина параллелізму в кожній ітерації паралельної програми неоднакова. Це виключено в методі паралелепіпедів і ряді інших методів. Досить повний опис методів практичної розробки паралельних алгоритмів для початкового вивчення представлено у відомій книзі Ортеги [3].

5.2. Граф операції-операнди

Для опису існуючих інформаційних залежностей в обираних алгоритмах рішення завдань може бути використана модель у вигляді графа " операції – операнди ". Для зменшення складності викладеного матеріалу при побудові моделі передбачатиметься, що час виконання будь-яких обчислювальних операцій є однаковим і дорівнює 1 (у тих чи інших одиницях виміру). Крім того, приймається, що передача даних між обчислювальними пристроями виконується миттєво без будь-яких витрат часу (що може бути справедливо, наприклад, при наявності загальної пам'яті, що в паралельній обчислювальній системі).

Уявімо безліч операцій, виконуваних в досліджуваному алгоритмі рішення обчислювальної задачі, і існуючі між операціями інформаційні залежності у вигляді ациклического орієнтованого графа G = (V, R), де V = {1,..., | V |} є безліч вершин графа, представляють виконувані операції алгоритму, а R є безліч дуг графа (при цьому дуга r = (i, j) належить графу тільки в тому випадку, якщо операція j використовує результат виконання операції i). Для прикладу на рисунку 5.4 показаний граф алгоритму обчислення площі прямокутника, заданого координатами двох протилежних кутів.

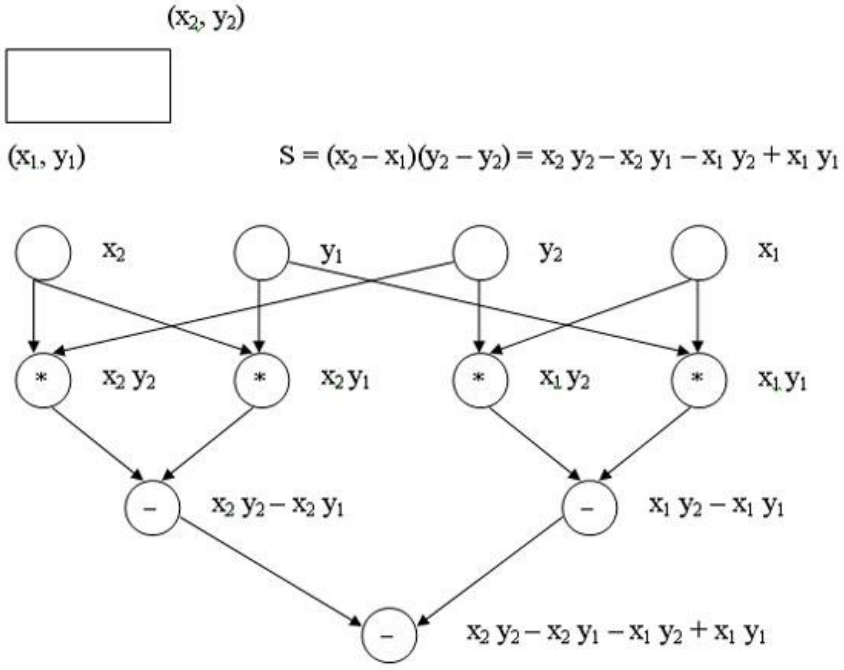


Рисунок 5.4. – Граф алгоритму обчислення площі прямокутника, заданого координатами двох протилежних кутів

Як можна помітити по наведеним прикладом, для виконання обраного алгоритму розв'язання задачі можуть бути використані різні схеми обчислень і побудовані відповідно різні обчислювальні моделі. Як буде показано далі, різні схеми обчислень володіють різними можливостями для розпаралелювання і, тим самим, при побудові моделі обчислень може бути поставлена задача вибору найбільш придатною для паралельного виконання обчислювальної схеми алгоритму.

У розглянутій обчислювальної моделі алгоритму вершини без вхідних дуг можуть використовуватися для завдання операцій введення, а вершини без вихідних дуг – для операцій виводу. Позначимо через V безліч вершин графа без вершин введення, а через d (G) – діаметр (довжину максимального шляху) графа.

Для паралельного програмування існує ряд мов. Основні з них:

– OpenMP – для багатопроцесорних систем із загальною пам'яттю.
– MPI – для багатопроцесорних систем з індивідуальною пам'яттю.

Існує часто деяка плутанина між OpenMP і MPI. ця плутанина цілком зрозуміло, бо є ще версія MPI- називається "Відкрити MPI ". З точ ки зору програміста, MPI є бібліотекою, яка містить процедури передачі повідомлень. OpenMP, з іншого боку, являє собою набір директив компілятора, які повідомляють OpenMP, якщо включений компілятор, які частини програми можуть бути запущені як нитки. Таким чином, різниця «нитки » проти « повідомлень». Давайте поглянемо на обидва методи.

5.3. Стандарт MPI

Найбільш поширеною бібліотекою паралельного програмування в моделі передачі повідомлень є MPI (Message Passing Interface). Рекомендованою безкоштовною реалізацією MPI є пакет MPICH, розроблений в Аргонській національні лабораторії.

MPI [8] є бібліотекою функцій міжпроцессорного обміну повідомленнями і містить близько 300 функцій, які поділяються на такі класи: операції точка-точка, операції колективного обміну, топологічні операції, системні та допоміжні операції. Оскільки MPI є стандартизований бібліотекою функцій, то написана із застосуванням MPI програма без переробок виконується на різних паралельних ЕОМ. Принципово для написання переважної більшості програм достатньо декілька функцій, які наведені нижче.

Функція MPI_Send є операцією точка-точка і використовується для посилання даних в конкретний процес.

Функція MPI_Recv також є точковою операцією і використовується для прийому даних від конкретного процесу. Для розсилки однакових даних всім іншим процесам використовується колективна операція MPI_BCAST, яку виконують всі процеси, як посилаючі, так і приймаючі. Функція колективного обміну MPI_REDUCE об'єднує елементи вхідного буфера кожного процесу в групі, використовуючи операцію *op*, і повертає об'єднане значення у вихідний буфер процесу з номером *root*.

- MPI_Send (адреса, count, тип даних, призначення, тег, акція);
- address – адреси посаних даних в буфері відправника;
- count – довжина повідомлення;
- datatype- тип посиланих даних;
- destination – ім'я процесу-одержувача;
- tag- для допоміжної інформації;
- comm – ім'я комунікатора;
- MPI_Recv (адреса, кількість, тип даних, джерело, тег, акція, статус);
- address – адреса одержуваних даних в буфері одержувача;
- count – довжина повідомлення;
- datatype-тип одержуваних даних;
- source – ім'я посилає процесу процесу;
- tag- для допоміжної інформації;
- comm – ім'я комунікатора;
- status – для допоміжної інформації;
- MPI_BCAST (адреса, count, тип даних, корінь, комм);
- root – номер розсилаючого (кореневого) процесу;
- MPI_REDUCE (sendbuf, recvbuf, граф, тип даних, оп, корінь, комм);
- sendbuf – адреса посилаючого буфера;
- recvbuf – адреса приймаючого буфера;
- count- кількість елементів у посилаючому буфері;
- datatype – тип даних;

– op – операція редукції;
– root – номер головного процесу.

Крім цього, використовується декілька організуючих функцій:

– MPI_INIT ();
– MPI_Comm_size (MPI_COMM_WORLD, numprocs);
– MPI_Comm_rank (MPI_COMM_WORLD, MyID);
– MPI_FINALIZE ().

Звернення до MPI_INIT присутнє в кожній MPI програмі і повинно бути першим MPI – зверненням. При цьому в кожному виконанні програми може виконуватися тільки один виклик, після виконання цього оператора всі процеси паралельної програми виконуються паралельно. MPI_INIT.

MPI_COMM_WORLD є початковим (і в більшості випадків єдиним) комунікатором і визначає комунікаційний контекст і пов'язану групу процесів. Звернення MPI_Comm_size повертає число процесів numprocs, запущених в цій програмі користувачем. Викликаючи MPI_Comm_rank, кожен процес визначає свій номер в групі процесів з деяким ім'ям. Рядок MPI_FINALIZE () має бути виконаний кожним процесом програми. Внаслідок цього ніякі MPI – оператори більше виконуватися не будуть.

5.4. MPI програма для обчислення числа π на мові С.

Для першої паралельної програми зручна програма обчислення числа π, оскільки в ній немає завантаження даних і легко перевірити відповідь. Обчислення зводяться до обчислення інтеграла за такою формулою:

$$Pi = \int_0^1 \frac{4}{1+x^2}dx \approx \frac{1}{n}\sum_1^n \frac{4}{1+x_i^2} \qquad (5.1)$$

де xi = (i-1/2) / n. Програма представлена рис.4.1.

Лістинг 5.1 – Програма обчислення числа π на мові С

```
#include "mpi.h"
#include <math.h>
int main (int argc, char ‘argv[])
{
int n, myid, numprocs, i; /’ число ординат, ім`я і число процесів’/
double PI25DT = 3.14159265358979323846264 3; /’використовується для оцінки точності обчислень ‘/
double mypi, pi, h, sum, x; /’ mypi – приватне значення п окремого процесу, pi – повне значення п ‘/
MPI_Init(&argc, &argv); /’ задаються системою’/
MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
while (1)
{
if (myid == 0) {
```

```
      printf ("Enter the number of intervals: (0 quits) ");
/'введення числа оординат'/
      scanf ("%d", &n);
      }
      MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
      if (n == 0) /' задається умова виходу з програми `/
      break;
      else {
      h = 1.0/ (double) n; обчислення приватного значення п деякого
процесу `/
      sum = 0.0;
      for (i = myid +1; i <= n; i+= numprocs) {
      x = h ` ((double)i - 0.5);
      sum += (4.0 / (1.0 + x'x));
      }
      mypi = h ` sum; /' обчислення приватного значення п деякого
процесу `/
      MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
      MPI_COMM_WORLD); /' збір повного значення п `/
      if (myid == 0) /'оцінка похибки обчислень `/
      printf ("pi is approximately %.16f. Error is
      %.16f\n", pi, fabs(pi - PI25DT));
      }
      }
      MPI_Finalize(); /' вихід з MPI `/
      return 0;
      }
```

5.5. Програма множення матриці на вектор

Результатом множення матриці на вектор є вектор результату. Для вирішення завдання використовується алгоритм, в якому один процес (головний) координує роботу інших процесів (підлеглих). Для наочності єдина програма матрично – векторного множення розбита на три частини: загальну частину (Лістинг 5.2), код головного процесу (Лістинг 5.3) і код підлеглого процесу (Лістинг 5.4).

У загальній частині програми описуються основні об'єкти завдання: матриця A, вектор b, результуючий вектор c, визначається число процесів (не менше двох). Завдання розбивається на дві частини: головний процес (master) і підпорядковані процеси. У задачі множення матриці на вектор одиниця роботи, коору потрібно роздати процесам, складається з скалярного відтворення рядка матриці A на вектор b. Знаком ! відзначені коментарі.

Лістинг 5.2 – Програма множення матриці на вектор: загальна частина

```
program main
use mpi
integer MAX_ROWS, MAX_COLS, rows, cols
parameter (MAX_ROWS = 1000, MAX_COLS = 1000)
! матриця_A, вектор b, результативний вектор c
```

```
      double precision a(MAX_ROWS,MAX_COLS), b(MAX_COLS),
c(MAX_ROWS)
      double precision buffer (MAX_COLS), ans /' ans - ім`я
результату'/
      integer myid, master, numprocs, ierr, status
(MPI_STATUS_SIZE)
      integer i, j, numsent, sender, anstype, row /' numsent -
число посланих стрічок,
      sender - ім'я процессу-відправника, anstype - номер посланої
стрічки'/
      call MPI_INIT(ierr)
      call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
      call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)
      ! головний процес - master
      master = 0
      ! кількість рядків і стовбців матриці A
      rows = 100
      cols = 100
      if (myid.eq. master) then
      ! код головного процесу
      else
      ! код підлеглого процесу
      endif
      call MPI_FINALIZE(ierr)
      stop
      end
```

Код головного процесу представлений на лістингу. 5.3. Одиницею роботи підпорядкованого процесу є множення рядка матриці на вектор.

Лістинг 5.3 – Програма для множення матриці на вектор: код головного процесу

```
      ! ініціалізація A і b
      do 20 j = 1, cols
      b(j) = j
      do 10 i = 1, rows
      a(i,j) = i
      10 continue
      20 continue
      numsent = 0
      ! посилання b кожному підлеглому процесу
      call MPI_BCAST(b, cols, MPI_DOUBLE_PRECISION, master,
      MPI_COMM_WORLD, ierr)
      !   посилання стрічки кожному підлеглому процесу; в TAG номер
стрічки= i
      do 40 i = 1,min(numprocs-l, rows)
      do 30 j = I, cols
      buffer(j) = a(i,j)
      30 continue
      call MPI_SEND(buffer, cols, MPI_DOUBLE_PRECISION, i, i,
      MPI_COMM_WORLD, ierr)
      numsent = numsent + l
```

```
40 continue
! прийом результату від підлеглого процесу
do 70 i = 1, rows
! MPI_ANY_TAG – вказує, що приймається будь-яка стрічка
call MPI_RECV(ans, 1, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)
sender = status (MPI_SOURCE)
anstype = status (MPI_TAG)
! визначаємо номер стрічки
c(anstype) = ans
if (numsent.lt. rows) then
!посилання наступної стрічки
do 50 j = 1, cols
buffer(j) = a(numsent+l, j)
50 continue
call MPI_SEND (buffer, cols, MPI_DOUBLE_PRECISION, sender,
numsent+l, MPI_COMM_WORLD, ierr)
numsent = numsent+l
else
! посилання признаку кінця роботи
call MPI_SEND(MPI_BOTTQM, 0, MPI_DOUBLE_PRECISION,sender,
0, MPI_COMM_WORLD, ierr)
endif
70 continue
```

Спочатку головний процес передає вектор b в кожен підлеглий процес, потім пересилає один рядок матриці в кожен підлеглий процес.

Головний процес, отримуючи результат від чергового підлеглого процесу, передє йому нову роботу. Цикл закінчується, коли всі рядки будуть роздані і будуть отримані результати.

При передачі даних з головного процесу в параметрі tag вказується номер переданого рядка. Цей номер після обчислення добутку разом з результатом буде відправлений у головний процес, щоб головний процес знав, де розміщувати результат.

Підлеглі процеси посилають результати в головний і параметр MPI_ANY_TAG в операції прийому головного процесу вказує, що головний він приймає рядки в будь-якій послідовності.

Параметр status забезпечує інформацію, що відноситься до отриманого повідомлення. У мові Fortran це – масив цілих чисел розміру MPI_STATUS_SIZE. Аргумент SOURCE містить номер процесу, який надіслав повідомлення, з цієї адреси, головний процес буде пересилати нову роботу. Аргумент TAG зберігає номер обробленого рядка, що забезпечує розміщення отриманого результату. Після того як головний процес розіслав всі рядки матриці A, на запити підлеглих процесів він відповідає повідомленням з відміткою 0.

Код підлеглого процесу представлений на лістингу 5.4.

Лістинг 5.4 – Програма для матрично-векторного множення: підлеглих процесів.

```
! прийом вектора b усіма підлеглими процесами
call MPI_BCAST(b, cols, MPI_DOUBLE_PRECISION, master,
MPI_COMM_WORLD, ierr)
!вихід, якщо процесів більше, ніж рядків матриці
if (numprocs.gt. rows) goto 200
! прийом рядка матриці
90 call MPI_RECV(buffer, cols, MPI_DOUBLE_PRECISION, master,
MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)
if (status (MPI_TAG).eq. 0) then go to 200
! кінець роботи
else
row = status (MPI_TAG)
! номер отриманого рядка
ans = 0.0
do 100 i = 1, cols
! скалярний добуток векторів
ans = ans+buffer(i)'b(i)
100 continue
! передача результату головному процесу
call MPI_SEND(ans,1,MPI_DOUBLE_PRECISION,master,row,
MPI_COMM_WORLD, ierr)
go to 90
! цикл для прийому наступного рядка матриці
endif
200 continue
```

Кожний підлеглий процес отримує вектор b. Після цього організується цикл, в такому стані, що підлеглий процес отримує черговий рядок матриці A, формує скалярний добуток рядка і вектора b, посилає результат головному процесу процесу, отримує новий рядок і.т.д.

Приклади задач на MPI представлені в [6,8,11]. Там також приводится повне описання бібліотеки MPI, методики використання MPI в средовищі мов C, C++ і Fortran. Описані методи побудови кластерів на ПЭВМ.

5.6. OpenMP

Інтерфейс OpenMP [14] задуманий як стандарт для програмування на масштабованих SMP- системах (модель загальної пам'яті). У стандарт OpenMP входять специфікації набору директив компілятора, процедур і змінних середовища. До появи OpenMP не було підходящого стандарту для ефективного програмування на SMP- системах.

Найбільш гнучким, загальноприйнятим інтерфейсом паралельного програмування є MPI (інтерфейс передачі повідомлень). однак модель передачі повідомлень:

– недостатньо ефективна на SMP- системах;

– складна в освоєнні, так як вимагає мислення в " не обчислювальних " термінах. POSIX -інтерфейс для організації ниток (Pthreads) підтримується широко (практично на всіх UNIX- системах), проте за багатьох причин не

підходить для практичного паралельного програмування: надто низький рівень, немає підтримки паралелізму за даними.

OpenMP можна розглядати як високорівневу надбудову над Pthreads (чи аналогічними бібліотеками ниток). За рахунок ідеї " інкрементного розпаралелювання " OpenMP ідеально підходить для розробників, бажаючих швидко розпаралелити свої обчислювальні програми з великими паралельними циклами. Розробник не створює нову паралельну програму, а просто послідовно додає в текст послідовної програми OpenMP – директиви. При цьому, OpenMP – досить гнучкий механізм, надає розробнику великі можливості контролю над поведінкою паралельного додатку. Передбачається, що OpenMP -програма на однопроцесорній платформі може бути використана як послідовної програми, тобто немає необхідності підтримувати послідовну і паралельну версії. Директиви OpenMP просто ігноруються послідовним компілятором.

Специфікація OpenMP для C / C + +, містить наступну функціональність:

– Директиви OpenMP починаються з комбінації символів "# pragma omp ". Директиви можна розділити на 3 категорії: визначення паралельної секції, поділ роботи, синхронізація. кожна директива може мати кілька додаткових:

– Компілятор з підтримкою OpenMP визначає макрос " _OPENMP ", котрий може використовуватися для умовної компіляції окремих блоків, характерних для паралельної версії програми.

– Розпаралелювання застосовується до for – циклів, для цього використовується директиви "# pragma omp for ". У паралельних циклах забороняється використовувати оператор break.

– Статичні (static) змінні, визначені в паралельній області програми, є загальними (shared).

– Пам'ять, виділена за допомогою malloc (), є спільною (однак вказівник на неї може бути як загальним, так і приватним).

– Типи і функції OpenMP визначені під включаємим файлом <omp.h>.

– Крім звичайних, можливі також "вкладені " (nested) м'ютекси – замість логічних змінних використовуються цілі числа, і нитка, вже захопивша м'ютекс, при повторному захопленні може збільшити це число.

Програмна модель OpenMP являє собою fork – join паралелізм, в якому головний потік по необхідності породжує групи потоків, при входженні програми в паралельні області програми.

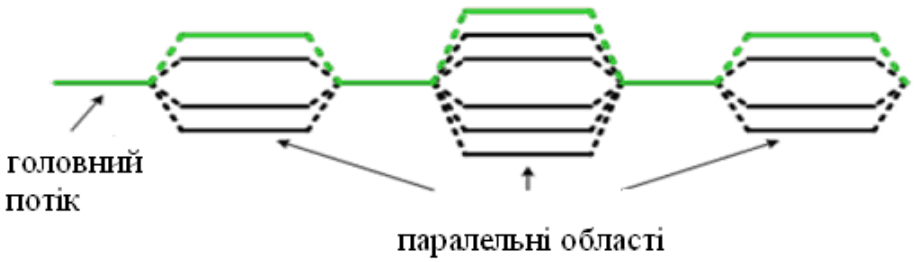


Рисунок 5.5. – Програмна модель OpenMP

У разі симетричного мультипроцесингу SMP на всіх процесорах процесорної системи виконується один примірник операційної системи, котра відповідає за розподіл прикладних процесів (завдань, потоків) між окремими процесорами.

Інтерфейс OpenMP є стандартом для програмування на масштабуємих SMP- системах з роздільною пам'яттю. У стандарт OpenMP входять опис набору директив компілятора, змінних середовища і процедур. За рахунок ідеї " інкрементального розпаралелювання " OpenMP ідеально підходить для розробників, які бажають швидко розпаралелити свої обчислювальні програми з великими паралельними циклами. Розробник не створює нову паралельну програму, а просто додає в текст послідовної програми OpenMP – директиви.

Передбачається, що OpenMP -програма на однопроцесорній платформі може бути використана в якості послідовної програми, тобто немає необхідності підтримувати послідовну і паралельну версії. директиви OpenMP просто ігноруються послідовним компілятором, а для виклику процедур OpenMP можуть бути підставлені заглушки, текст яких наведений у специфікаціях. У OpenMP будь-який процес складається з декількох ниток управління, які мають загальний адресний простір, але різні потоки команд і роздільні стеки. У простому випадку, процес складається з одної нитки.

Зазвичай для демонстрації паралельних обчислень використовують просту програму обчислення числа π. Розглянемо, як можна написати таку програму в OpenMP. Число π можна визначити наступним чином:

$$\int_0^1 \frac{1}{1+x^2}dx = \text{arctg}(1)\text{-arctg}(0)= \pi/4. \qquad (5.2)$$

Обчислення інтеграла потім замінюють обчисленням суми:

$$\int_0^1 \frac{4}{1+x^2}dx = \frac{1}{n}\sum_{i=1}^{n}\frac{4}{1+x_i^{\,2}}, \quad \text{где: } x_i = \frac{1}{n}\cdot\left(i-\frac{1}{2}\right) \qquad (5.3)$$

У послідовну програму вставлені два рядки (директиви), і вона стає паралельною (Лістинг 5.5).

Програма починається як єдиний процес на головному процесорі.

Він виконує всі оператори аж до першої конструкції типу # pragma omp.

У розглянутому прикладі це оператор parallel for, при виконанні якого породжується безліч процесів з відповідним кожному процесу оточенням.

У разі симетричного мультіпроцесінгу SMP на всіх процесорах процесорної системи виконується один примірник операційної системи, котра відповідає за розподіл прикладних процесів (завдань, потоків) між окремими процесорами. Розпаралелювання застосовується до for – циклів, для цього використовується директива "# pragma omp for ", за якою ОС роздає процесорам (ядрам) особистий примірник програми, як в SPMD, попросту передає один і той самий відрізок програми на задане число процесорів. Розпаралелювання застосовується до for – циклів, для цього використовується

директива "# pragma omp for ", по котрі ОС роздає процесорам (ядрам) особистий екземпляр програми, як в SPMD.

Лістинг 5.5 – Обчислення числа пі на мові С.

```
#include <stdio.h>
double f(double y) {return(4.0/(1.0+y*y));}
int main()
{
double w, x, sum, pi;
int i;
int n = 1000000;
w = 1.0/n;
sum = 0.0;
#pragma omp parallel for private(x) shared(w)\
reduction(+:sum)
for(i=0; i < n; i++)
{
x = w*(i-0.5);
sum = sum + f(x);
}
pi = w*sum;
printf("pi = %f\n", pi);
}
```

У розглянутому прикладі оточення складається з локальної (PRIVATE) змінної x, змінної sum редукції (REDUCTION) і однієї роздільної (SHARED) змінної w. Змінні x і sum локальні в кожному процесі без поділу між декількома процесами. Мінлива w розташовується в головному процесі. Оператор REDUCTION має в якості атрибута операцію, яка застосовується до локальних копій паралельних процесів в кінці кожного процесу для обчислення значення змінної в головному процесі.

Змінна циклу і є локальною в кожному процесі, оскільки саме з унікальним значенням цієї змінної породжується кожен процес. Параллельно процеси завершуються оператором END DO, виступаючим як синхронізуючий бар'єр для породжених процесів. Після завершення всіх процесів триває тільки головний процес.

Директиви OpenMP з точки зору С є коментарями і починаючинаються з комбінації символів # pragma, тому наведена вище програма може без змін виконуватися на послідовній ЕОМ у звичайному режимі.

Розпаралелювання в OpenMP виконується явно за допомогою вставки в текст програми спеціальних директив, а також виклику допоміжних функцій. При використанні OpenMP передбачається SPMD -модель (Single Program

Multiple Data) паралельного програмування, в рамках якої для всіх паралельних ниток використовується один і той же код.

Програма починається з послідовної області – спочатку працює один процес (нитка), при вході в паралельну область породжується (компілятором) ще деяке число процесів, між якими надалі розприділяються частини коду. По завершенні паралельної області всі нитки, крім однієї (нитки майстра), завершуються, і починається послідовна область. У програмі може бути будь-яка кількість паралельних і послідовних областей.

Крім того, паралельні області можуть бути також вкладеними одина в одну. На відміну від повноцінних процесів, породження ниток є відносно швидкою операцією, тому часті породження і завершення ниток не так сильно впливають на час виконання програми.

Після отримання виконуваного файлу необхідно запустити його на потрібні кількості процесорів. Для цього зазвичай потрібно задати кількість ниток, що виконують паралельні області програми, визначивши значення змінної середовища MP_NUM_THREADS. Після запуску починає працювати одна нитка, а всередині паралельних областей одна і та ж програма буде виконуватись усім набором ниток. При виході з паралельної області виробляється неявна синхронізація і знищуються всі нитки, крім первинної. Все породжені нитки виконують один і той же код, відповідної паралельної області. Передбачається, що в SMP- системі нитки будуть розподілені по різноманітних процесорах, однак це, як правило, знаходиться у веденні операційної системи.

Перед запуском програми кількість ниток, що виконують паралельну область, можна задати, за допомогою змінної середовища MP_NUM_THREADS.

Наприклад, в Linux це можна зробити за допомогою наступної команди: export OMP_NUM_THREADS = n. Функція omp_get_num_procs () повертає кількість процесорів, доступних для використання програмі користувача на момент виклику. Директиви master (master... end master) виділяють ділянку коду, який буде виконаний тільки ниткою – майстром. Решта нитки просто пропускають дану ділянку і продовжують роботу з оператора, розміщеного слідом за ним. Неявній синхронізації дана директива передбачає.

Модель даних в OpenMP припускає наявність як загальної для всіх ниток області пам'яті, так і локальної області пам'яті для кожної нитки. У OpenMP змінні в паралельних областях програми поділяються на два основних класи:

- shared (загальні; всі нитки бачать одну й ту ж змінну);
- private (локальні, кожна нитка бачить свій екземпляр змінної).

Загальна змінна завжди існує лише в одному екземплярі для всієї області дії. Оголошення локальної змінної викликає породження свого примірника даної змінної (того ж типу і розміру) для кожної нитки.

Зміна ниткою значення своєї локальної змінної ніяк не впливає на зміну значення цієї ж локальної змінної в інших нитках.

Якщо декілька змінних одночасно записують значення загальної змінної без виконання синхронізації, або якщо як мінімум одна нитка читає значення загальної змінної і як мінімум одна нитка записує значення цієї змінної без

виконання синхронізації, то виникає ситуація так званої «гонки даних». Для синхронізації використовується оператор barrier:

```
# pragma omp barrier
```

Нитки, виконують поточну паралельну область, дійшовши до цієї директиви, чекають, поки всі нитки не дійдуть до цієї точки прграмми, після чого розблоковуються і продовжують працювати далі.

Директиви OpenMP просто ігноруються послідовним компілятором, а для виклику функцій OpenMP можуть бути підставлені спеціальні «заглушки » (stub), текст яких наведений в описі стандарту. Вони гарантують коректну роботу програми в послідовному випадку – потрібно тільки перекомпілювати програму і підключити іншу бібліотеку.

OpenMP може використовуватися спільно з іншими технологіями паралельного програмування, наприклад, з MPI. Звичайно в цьому випадку MPI використовується для розподілу роботи між декількома обчислювальними вузлами, а OpenMP потім використовується для розпаралелювання на одному вузлі.

Лістинг 5.6 – Перемноження матриць на мові С.

```
#include <stdio.h>
#include <omp.h>
#define N 4096
double a[N][N], b[N][N], c[N][N];
int main()
{
int i, j, k;
double t1, t2;
// ініціалізація матриць
for (i=0; i<N; i++)
for (j=0; j<N; j++)
a[i][j]=b[i][j]=i’j;
t1=omp_get_wtime();
// основний обчислювальний блок
#pragma omp parallel for shared(a, b, c) private(i, j,
k)
for(i=0; i<N; i++){
for(j=0; j<N; j++){
c[i][j] = 0.0;
for(k=0; k<N; k++) c[i][j]+=a[i][k]’b[k][j];
}
}
t2=omp_get_wtime();
printf("Time=%lf\n", t2-t1);
}
```

Найпростіша програма, що реалізує множення двох квадратних матриць, представлена на лістингу 5.6. У програмі заміряється час на основний обчислювальний блок, що не включає початкову ініціалізацію. В основному обчислювальному блоці програми мовою Фортран змінено порядок циклів з параметрами *i* і *j* для кращої відповідності правилам розміщення елементів масивів.

ЛЕКЦІЯ 6. ПОНЯТТЯ ПРОМІЖНОГО СЕРЕДОВИЩА (MIDDLEWARE) ДЛЯ GRID. ПРОЦЕС ВИКОНАННЯ ЗАВДАННЯ GRID. ПІДХОДИ ДО ОРГАНІЗАЦІЇ СКЛАДНИХ СЕРВІСІВ ТА ПОТОКІВ РОБІТ. ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ ТА GRID. ЗАДАЧІ В GRID ТА ОСНОВНІ ОПЕРАЦІЇ НАД НИМИ. КОМПОНУВАННЯ СКЛАДНИХ ЗАДАЧ, ПОТОКИ ЗАДАЧ.

6.1. Поняття проміжного середовища (middleware) для GRID

Проміжні́ програ́мні за́соби (також Сполучні́ програ́мні за́соби) англ. middleware – це програмні засоби, що пов'язують програмні компоненти або прикладні програми.

Програмні засоби складаються з набору послуг, що дозволяють декільком процесам, запущених на одній або декількох машинах взаємодіяти через мережу. Ця технологія забезпечує взаємодію при переході до розподілених архітектур, які використовуються найчастіше для підтримки та спрощення складних, розподілених прикладних програм. До проміжних програмних засобів відносять веб-сервери, сервери програм та інші аналогічні інструменти, які підтримують розробку та впровадження програм. Проміжні програмні засоби є невід'ємною частиною сучасних інформаційних технологій, заснованих на XML, SOAP, Web Service та сервісно-орієнтованої архітектури.

Такі програмні засоби розташовані між прикладними програмами, що працюють на різних операційних системах. Вони схожі на середній шар трирівневої архітектури, що знаходиться на одній системі, за винятком того, що вона розтягується на кілька систем або програм. Прикладами можуть бути бази даних, телекомунікаційні програмні засоби, монітори транзакцій, а також програмні засоби повідомлень і черг.

Відмінність між системними і проміжними програмними засобами, не дуже чітка. Якщо основна функціональність ядра надається самою операційною системою, то деякі функції, що раніше надавалися лише окремим проміжними програмними засобами тепер інтегровані в операційні системи. Типовим прикладом є стек TCP/IP в телекомунікаціях, який сьогодні включено практично у всі операційні системи.

У технологіях моделювання проміжні програмні засоби за звичай використовуються в контексті HLA, що застосовується в багатьох розподілених моделювання. Це – прошарок програмних засобів, що знаходиться між прикладними програмами і інфраструктурою часу виконання.

ObjectWeb визначає проміжні програмні засоби, як: «Шар програмних засобів, що знаходиться між ОС і прикладними програмами, на кожній стороні розподіленої обчислювальної системи в мережі».

Проміжні програмні засоби є відносно новим застосунком для обчислювальної галузі. Вони здобули популярність в 1980-ті роки в якості вирішення проблеми поєднання старих успадкованих систем з новими архітектурами, хоча сам термін вживався з 1968 року. [2] Він також сприяв

розподіленим обчисленням, підключення декількох застосунків для створення більших програм, як правило, через мережу.

IBM, Red Hat і Oracle є основними постачальниками проміжних програмних засобів. Такі виробники, як SAP AG, TIBCO, Mercator Software, Crossflo, Vitria та webMethods були спеціально створені для забезпечення веб-інструменти проміжних програмних засобів. Також існують групи, як Apache Software Foundation і консорціум ObjectWeb, що заохочують розвиток відкритих проміжних програмних засобів.

На відміну від операційних системам та мережевих служб, сервіс проміжних ПЗ має більш функціональний прикладний програмний інтерфейс, що надає програмі:

- Прозорий пошук в мережі, дозволяючи взаємодію з іншими програмами або службами.
- Бути незалежною від мережевих служб.
- Бути завжди надійною і доступною.

Система класифікації Гурвіца (Hurwitz) організує багато видів проміжних ПЗ. Ці класифікації, засновані на масштабовності і можливості відновлення:

- Виклик віддалених процедур – клієнт робить виклики процедур, що працюють на віддалених системах. Виклики можуть бути синхронні або асинхронні.
- Проміжне ПЗ побудоване на повідомленнях – повідомлення, відправлені клієнту збираються і зберігаються до тих пір, поки з’явиться можливість їх обробити, клієнт продовжує працювати з іншими даними.
- Брокер об'єктних запитів – цей тип проміжних ПЗ дозволяє програмам передавати об’єкти і запити в об’єктно-орієнтованій системі.
- Доступ даних на базі SQL – проміжне ПЗ між програмами та серверами баз даних.
- Вбудовані проміжні ПЗ – ПЗ та вбудовані програми послуг зв'язку та інтерфейсу інтеграції, що працюють між вбудованими програмами та операційними системами реального часу.

Інші джерела містять також такі додаткові класифікації:

- Монітори обробки транзакцій – надає інструменти та середовище для розробки та впровадження розподілених програм [4]
- Сервер програм – програмні засоби, встановлені на комп’ютері для спрощення виконання інший програм.
- Шина корпоративної служби – шар абстрагування поверх корпоративної системи обміну повідомленнями.

6.2. Процес виконання завдання GRID

Під англійським терміном GRID розуміється сукупність просторово розподілених обчислювальних вузлів, пов'язаних деякою мережею для обміну даними. Надалі замість GRID буде використовуватися слово Грід. Це слово не абревіатура, і в прямому перекладі означає «грати», але правильніше його

вживати в сенсі «обчислювальна мережа» за аналогією з енергомережею, з якої можна споживати енергію, не піклуючись про те, ким і де вона вироблена. Різниця між Web Service і Грід полягає в наступному:

– Web Service дозволяє клієнту виконати на обладнанні власника ресурсу деяку функцію зі списку, складеного власником цього обладнання.

– Грід – метод використання глобально процесорних потужностей і систем зберігання інформації (дискові системи великої ємності) на основі погодинної оренди без їх фізичного переміщення в просторі. Елементами Грід в основному є кластери, а не окремі комп'ютери.

Природно, Грід включає все, напрацьоване в WebServices. Більш того, протоколи WebServices (WSDL, SOAP, UDDI), засоби адресації в розширеному варіанті є основними протоколами. Основна інформація з Globus GRID представлена в [16].

Однією з причин створення европейского Грід стала необхідність обробки величезного обсягу інформації, яка надходить з Великого адронного коллайдера (ВАК), створеного ЦЕРНІ досліджень. Для нього використаний 27-кілометровий підземний тунель, прокладений на глибині близько 100 метрів на кордоні Швейцарії та Франції. БАК призначений для розгону протонів і важких іонів, які при зіткненні на зустрічних пучках породжують нові частинки, вивчення цих частинок сприятиме вивченню основ світобудови. З БАК за рік надходитиме 10 Петабайт даних. Для обробки цього гігантського обсягу даних буде використовується технологія розподілених обчислень Грід.

Загальна структура Грід на прикладі одного вузла представлена на рис. 6.1.

Клієнт звертається до реєстру ресурсів MDS (Monitoring and Discovery Service), щоб отримати відомості про наявність і місце розташування потрібного ресурсу. Потім клієнт звертається до ресурсу, щоб отримати інтерфейс ресурсу, в якому вказується спосіб завдання ресурсу необхідної роботи. Нарешті, клієнт передає ресурсу завдання на мові RSL (Resource Specification Language) і отримує результат.

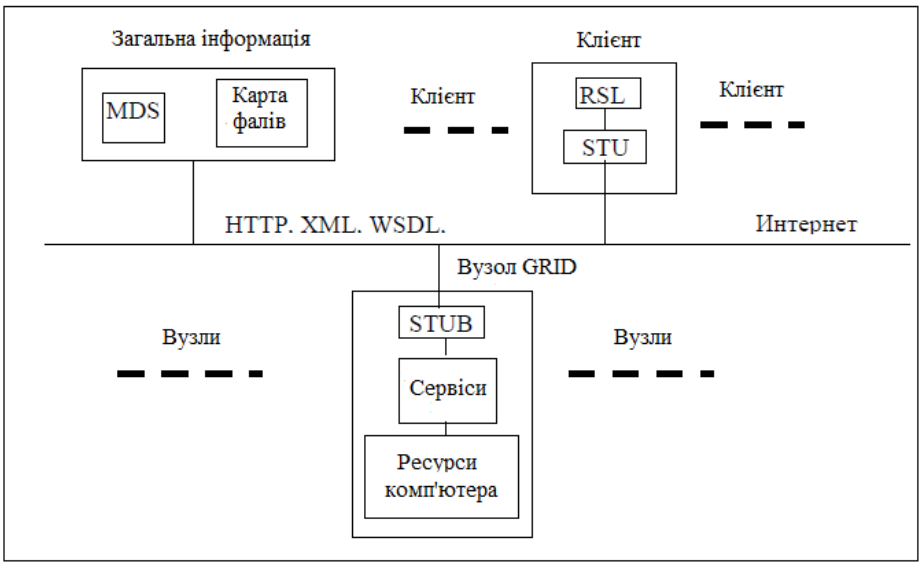


Рисунок 6.1. – Організація ГРІД

У глобальній мережі знаходяться десятки тисяч (і більше) вузлів, кожен з них може бути як ресурсом, так і клієнтом. Крім клієнтів і ресурсів є загальні служби, наприклад, довідкова система і карта розташування файлів. Головне полягає в тому, що на каналах обміну використовується стандарт Web-служб: XML, WSDL, SOAP та ін

У середовищі Грід присутні такі елементи:

- Програми користувача.
- Ресурси (апаратура, ОС, кластерне ПЗ і т.п.).
- Проміжне програмне забезпечення (Middleware), яка виступає в ролі посередника між одними програмами та ресурсами.

Middleware включає великий обсяг прогамного забезпечення, розробляється великими організаціями і строго стандартизуется, щоб забезпечити взаємно перехресне використання частин цього Middleware різними розробниками.

У число найбільш відомих пакетів middleware входять:

- GT4 – розроблений в США за проектом Globus. Він буде розглянутий далі.
- gLITE. Його умовно можна назвати європейським проектом, оскільки його розробка курується ЦЕРН.

6.2.1. Паралельне обчислення в ГРІД. Пакет G2

Для виконання паралельних обчислень на кластерах використовується бібліотека MPI. Пакет MPICH-G2 [19] є Грід орієнтована повна реалізація стандарту MPI-1, яка використовує служби Globus Toolkit для прозорої роботи в середовищі Грід, в тому числі і гетерогенної.

MPICH -G2:

- дозволяє програмісту з'єднувати комп'ютерів різної архітектури;
- планує розподіл ресурсів;
- автоматично перетворює дані в повідомленнях між комп'ютерами різної архітектури;
- звільняє користувача від роботи з вивчення специфіки конкретних машин і дозволяє користувачеві запускати багатопроцесорне додаток однією командою mpirun.

Схема виконання обчислень на MPI відповідає рис.6.6, але в якості планувальника co-allocator використовується програма DUROC яка:

- З усіх можливих ресурсів вибирає тільки такі, які можуть працювати з MPI.
- Вибрані ресурси рознесені в просторі, розрізняються за характеристиками (швидкодія процесорів, обсяг пам'яті, швидкість передачі даних по каналах зв'язку та ін), тому часи виконання локальних обчислень можуть виявитися різними. Отже, неоходімо встановити факт закінчення всіх локальних робіт і обмінів даними, перш ніж запускати наступні відрізки локальних обчислень. Це досягається за допомогою операторів синхронізації, наприклад, MPI_Barrier.

– Оптимізує вибір шляхів обміну даними з урахуванням їх пропускної спроможності і послідовності обмінів, щоб скоротити загальний час передачі даних. Канали відрізняються по пропускній здатності: intramachine messaging, short (LAN) and long (WAN). З цієї стратегії MPICH -G2 впорядковує різні комунікаційні методи на наступних припущеннях:

```
WAN-TCP < LAN-TCP < intra-machine TCP < vendor-supplied MPI
```

Порядок запуску прокладання такий:

– Щоб запустити MPICH -G2 додаток, користувачеві потрібно отримати відкритий ключ, який використовується для аутентифікації користувача на кожному віддаленому сайті.

– Коли ідентифікація проведена, користувач використовує стандартну mpirun команду, щоб запросити створення MPI обчислень. Реалізація mpirun використовує скрипти RSL, які пишуть користувачі. У них ідентифікуються ресурси (тобто комп'ютери), описують вимоги (кількість CPU, пам'яті, необхідного часу і т. д.) і параметри (розміщення програм, аргументи командного рядка, змінні середовища та т.д.) для кожного ресурсу.

– На основі цієї інформації MPICH-G2 викликає DUROC, щоб спланувати і запустити додаток на різних комп'ютерах. DUROC реалізує операцію розміщення через множинні RM (ресурсні менеджери) в рамках Globus.

– Для кожного подвичісленія DUROC генерує GRAM запит до віддаленого GRAM серверу, який розпізнає користувача, виконує локальну авторизацію і потім взаємодіє з локальним планувальником, щоб ініціювати обчислення. DUROC і пов'язані з MPICH-G2 бібліотеки пов'язують різні подвичісленія в єдине MPI обчислення.

– DUROC і GRAM також взаємодіють при моніторингу та виконанні додатки. Кожен GRAM сервер стежить за життєвим циклом його обчислень на всіх стадіях: затримка, виконання, закінчення.

6.2.2. Пакет gLite

gLite – європейський пакет middleware для Грід. Основна відмінність пакета gLite від GT4 полягає в тому, що крім інструментальних засобів, в нього входить ширший набір служб. gLite істотно спирається на досвід низки великих європейських проектів і створюється колективно – в його розробці беруть участь багато дослідницьких центрів Європи. Все це проекти, включаючи gLite, мають багато спільного, оскільки в тій чи іншій мірі засновані на одній базі – системі Globus Toolkit. gLite є основою проекту EGEE (Enabling Grids for E – SciencE – "Розгортання грід для е- науки").

ЛЕКЦІЯ 7. АРХІТЕКТУРНІ РІВНІ ОБЧИСЛЮВАЛЬНОЇ ХМАРИ. ІНФРАСТРУКТУРА ЯК СЕРВІС. ПЛАТФОРМА ЯК СЕРВІС. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЯК СЕРВІС

Мета даної лекції – отримати відомості про появу хмарних обчислень, їх переваги та недоліки.

Рисунок 7.1. – Приклади застосування концепції SaaS

Перші ідеї про використання обчислень як публічної послуги були запропоновані ще в 1960 – х відомим вченим в галузі інформаційних технологій, винахідником мови Lisp, професором MIT і Стенфордського університету Джоном Маккарті (John McCarthy). Реалізація першого реального проекту приписується компанії Salesforce.com, заснованої в 1999 році. Саме тоді і з'явилося перше речення нового виду b2b продукту «Програмне забезпечення як сервіс» (" Software as a Service ", " SaaS "). Певний успіх Salesforce в цій області порушив інтерес у гігантів IT індустрії, які поспіхом повідомили про свої дослідження в області хмарних технологій. І ось вже перше бізнес – рішення під назвою «Amazon Web Services» було запущено в 2005 році компанією Amazon.com, яка з часів кризи доткомів активно займалася модернізацією своїх датацентрів. Наступним свою технологію поступово ввела Google, почавши з 2006 року b2b пропозицію SaaS сервісів під назвою «Google Apps». І, нарешті, свою пропозицію анонсувала компанія Microsoft, презентувавши її на конференції PDC 2008 під назвою «Azure Services Platform».

Рисунок – 7.2. SaaS сервіси Google

Сам факт високої зацікавленості найбільших гравців ринку IT демонструє певний статус хмарних обчислень як тренда 2009 -2010 років. Крім того, з релізом Microsoft Azure Service Platform безліч експертів пов'язує новий виток розвитку веб – технологій і вихід всієї сфери хмарних обчислень на новий рівень.

Нагадаємо, що під хмарними обчисленнями ми розуміємо програмно – апаратне забезпечення, доступне користувачеві через Інтернет або локальну мережу у вигляді сервісу, що дозволяє використовувати зручний інтерфейс для віддаленого доступу до виділених ресурсів (обчислювальних ресурсів, програм і даних).

На даний момент більшість хмарних інфраструктур розгорнуто на серверах датацентрів, використовуючи технології віртуалізації, що фактично дозволяє будь -якому призначеному для користувача додатком використовувати обчислювальні потужності, абсолютно не замислюючись про технологічні аспектах. Тоді можна розуміти «хмара» як єдиний доступ до обчислень з боку користувача.

7.1. Види хмарних обчислень

З поняттям хмарних обчислень часто пов'язують сервіси що мають (Everything as a service) технології, такі як:

– «Інфраструктура як сервіс» ("Infrastructure as a Service" або "IaaS")

– «Платформа як сервіс» ("Plaatform as a Service", "PaaS")

– «Програмне забезпечення як сервіс» ("Software as a Service" або "SaaS ").

Розглянемо кожну з цих технологій докладніше.

7.1.1. Інфраструктура як сервіс (IaaS)

IaaS – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

IaaS складається з трьох основних компонентів:

– Апаратні засоби (сервери, системи зберігання даних, клієнтські системи, мережеве обладнання).

– Операційні системи та системне ПЗ (засоби віртуалізації, автоматизації, основні засоби управління ресурсами).

– Сполучне ПО (наприклад, для управління системами).

IaaS заснована на технології віртуалізації, що дозволяє користувачу обладнання ділити його на частини, які відповідають поточним потребам бізнесу, тим самим збільшуючи ефективність використання наявних обчислювальних потужностей. Користувач (компанія або розробник ПЗ) повинен буде оплачувати всього лише реально необхідні йому для роботи серверний час, дисковий простір, мережеву пропускну спроможність та інші ресурси. Крім того, IaaS надає в розпорядження клієнта весь набір функцій управління в одній інтегрованої платформі.

Рисунок 7.3. – Компоненти хмарної інфраструктури

IaaS позбавляє підприємства від необхідності підтримки складних інфраструктур центрів обробки даних, клієнтських і мережевих інфраструктур, а також дозволяє зменшити пов'язані з цим капітальні витрати і поточні витрати. Крім того, можна отримати додаткову економію, при наданні послуги в рамках інфраструктури спільного використання.

Першопрохідцями в IaaS вважається компанія Amazon, які на сьогоднішній день пропонують два основних IaaS – продукту: EC2 (Elastic Compute Cloud) і S3 (Simple Storage Service). EC2 являє собою Xen -хостинг зі статичними VPS – характеристиками, що не розширюються на льоту (хоча багато подібні сервіси вже надають т.зв. auto scaling). Сховище S3 має інтерфейс WebDAV і підтримує роботу з багатьма відомими мовами програмування.

Серед інших інфра – сервісних компаній можна відзначити:

– GoGrid має дуже зручний інтерфейс для управління VPS, а також cloud storage з підтримкою протоколів SCP, FTP, SAMBA / CIFS, RSYNC, причому розмір сховища масштабується на льоту. Незабаром розробники обіцяють додати управління за допомогою API.

– Enomaly являє собою рішення для розгортання та управління віртуальними додатками в хмарі, при цьому управління послугами здійснюється через браузер. Приємним доповненням є автоматичне масштабування віртуальних машин під поточного навантаження, а також автобалансування навантаження. Серед підтримуваних віртуальних архітектур підтримуються Linux, Windows, Solaris і BSD Guests. Для віртуалізації застосовують не тільки Xen, але і KVM, а також VMware.

– Eucalyptus являє собою програмний комплекс з відкритим кодом для реалізації cloud computing на кластерних системах. В даний час інтерфейс сумісний з Amazon EC2, але заявлена підтримка та інших.

7.1.2. Платформа як сервіс (PaaS)

PaaS – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб-додатків як послуги.

Для розгортання веб-додатків розробнику не потрібно купувати обладнання та програмне забезпечення, немає необхідності організовувати їх підтримку. Доступ для клієнта може бути організований на умовах оренди.

Такий підхід має такі переваги:

- масштабованість;
- відмовостійкість;
- віртуалізація;
- безпеку.

Масштабованість PaaS передбачає автоматичне виділення і звільнення необхідних ресурсів залежно від кількості обслуговуваних додатком користувачів.

PaaS як інтегрована платформа для розробки, тестування, розгортання і підтримки веб-додатків дозволить весь перелік операцій з розробки, тестування, та розгортанню веб-додатків виконувати в одному інтегрованому середовищі, виключаючи тим самим витрати на підтримку окремих середовищ для окремих етапів.

Здатність створювати вихідний код і надавати його в загальний доступ всередині команди розробки значно підвищує продуктивність по створенню додатків на основі PaaS.

Найвідомішим прикладом такої платформи є AppEngine від Google, яка пропонує хостинг для веб-додатків з можливістю купувати додаткові обчислювальні ресурси (наприклад, для тестування високих навантажень). Для запуску додатків Google AppEngine на віртуальних кластерних системах була розроблена платформа AppScale, яка не має ніякого відношення до Google.

У системах веб-пошуку і контекстної реклами компанії Yahoo використовується платформа Hadoop, орієнтована на передачу великих обсягів даних між мережевими серверами. На базі Hadoop побудовані HBase (аналог бази даних Google BigTable), а також HDFS (Hadoop Distributed File System, аналог Google File System).

Ще одним яскравим представником PaaS є продукти компанії Mosso:

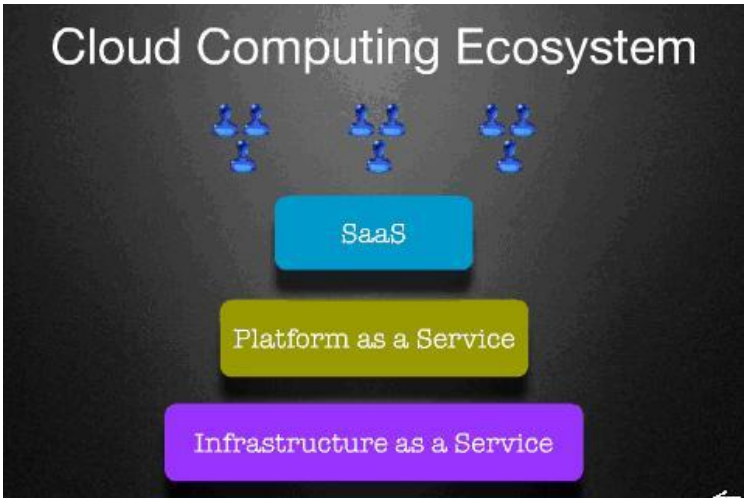
– Cloud Sites – веб-хостинг (Linux, Windows, Mail) для навантажувальних веб – проектів з можливістю розширювати базові безкоштовні – можливості за додаткову плату (трафік, сховище даних, обчислювальна потужність).

– Cloud Files – файловий cloud-хостинг з щомісячною погігабайтной оплатою за обсяг збережених файлів. Управління здійснюється через браузер, або за допомогою API (PHP, Python, Java, .NET, Ruby).

– Cloud Servers – погодинна оренда серверів (RAM на годину), з можливістю вибору серверної ОС. Можна змінювати характеристики сервера, але не в режимі реального часу. Незабаром розробники обіцяють зробити API для управління серверами.

Ну а в центрі всієї хмарної інфраструктури Microsoft – операційна система Windows Azure. Windows Azure створює єдине середовище, що включає хмарні аналоги серверних продуктів Microsoft (реляційна база даних SQL Azure, що є аналогом SQL Server, а також Exchange Online, SharePoint Online і Microsoft Dynamics CRM Online) і інструменти розробки (.NET Framework і Visual Studio, оснащена в версії 2010 набором Windows Azure Tools). Так, наприклад, програміст, що створює сайт в Visual Studio 2010, може не виходячи з програми розмістити свій сайт в Windows Azure.

7.1.3. Програмне забезпечення як сервіс (SaaS)



SaaS – модель розгортання програми, яка надає додаток кінцевому користувачеві як послугу на вимогу (on demand). Доступ до такого додатку здійснюється за допомогою мережі, а найчастіше за допомогою Інтернет-браузера.

У даному випадку, основна перевага моделі SaaS для клієнта полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання та програмного забезпечення, що працює на ньому. Цільова аудиторія – кінцеві споживачі.

У моделі SaaS:

– додаток пристосований для віддаленого використання;

– одним додатком можуть користуватися декілька клієнтів;

– оплата за послугу стягується або як щомісячна абонентська плата, або на основі сумарного обсягу транзакцій;

– підтримка програми входить вже до складу оплати;

– модернізація програми може проводитися обслуговуючим персоналом плавно і прозоро для клієнтів.

З точки зору розробників програмного забезпечення, модель SaaS дозволить ефективно боротися з неліцензійним використанням програмного забезпечення, завдяки тому, що клієнт не може зберігати, копіювати і встановлювати програмне забезпечення.

По-суті, програмне забезпечення в рамках SaaS можна розглядати в якості більш зручною і вигідною альтернативи внутрішнім інформаційним системам.

Розвитком логіки SaaS є концепція WaaS (Workplace as a Service – робоче місце як послуга). Тобто клієнт отримує в своє розпорядження повністю оснащене всім необхідним для роботи ПЗ віртуальне робоче місце.

За нещодавно опублікованими даними SoftCloud попитом користуються наступні SaaS додатки (у порядку убування популярності):

- Пошта.
- Комунікації (VoIP).
- Антиспам і антивірус.
- Helpdesk.
- Управління проектами.
- Дистанційне навчання.
- CRM.
- Зберігання і резервування даних.

Дуже схожими є продукти MobileMe (Apple), Azure (Microsoft) і LotusLive (IBM). Суть даних сервісів в тому, що вони надають користувачам доступ до зберігання своїх даних (контакти, пошта, файли), а також для спільної роботи декількох користувачів з документами.

Питаннями зберігання призначених для користувача даних в Інтернет спантеличена і компанія Google, яка розробляє проект GDrive, який буде представляти собою віртуальний жорсткий диск, який буде визначатися ОС як локальний. Також заявлено, що можна буде зберігати необмежену кількість даних, що звучить досить заманливо.

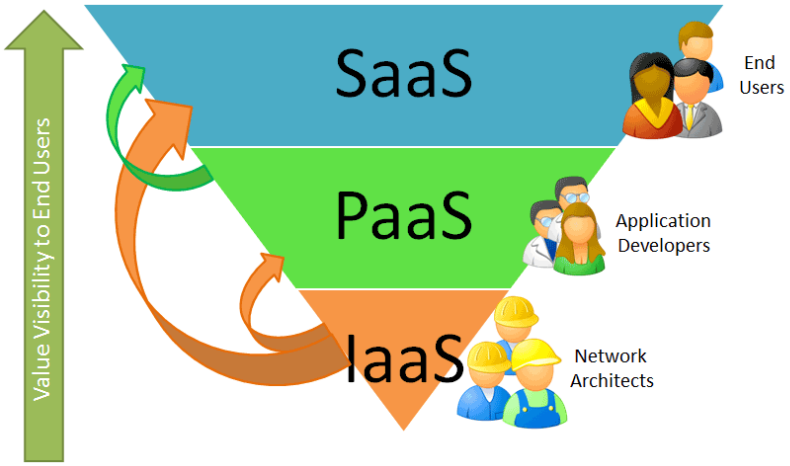


Рисунок 7.4. – Сервіси SaaS мають найбільшу споживчу базу

Зберігання файлів без обмежень також пропонує MediaFire.com. Існує як повністю безкоштовне використання (правда, з деякими обмеженнями, наприклад, на максимальний розмір файлу), так і покупка преміум-аккаунта, що розширює можливості (наприклад, шифрування файлів, отримання прямих посилань на скачування).

Ще одним цікавим представником виду SaaS є продукт iCloud, який представляє собою операційну систему, працювати з якою можна безпосередньо через браузер. Інтерфейс операційної системи виконаний в стилі Windows Vista/ XP. На сьогоднішній день проект знаходиться у стадії бети і в самій ОС реалізований мінімум додатків.

Також до SaaS належать послуги Online backup, або, простіше кажучи – резервного копіювання даних. Користувач просто платить абонентську плату, а сервіси самі автоматично в певний час шифрують дані з комп'ютера або іншого пристрою і відправляють їх на віддалений сервер, тим самим дані можуть бути доступні з будь-якої точки земної кулі. Дану послугу зараз надають безліч компаній, у тому числі, такі як Nero і Symantec.

Цікаве застосування cloud-технологій знайшли і розробники комп'ютерних ігор: тепер сучасним комп'ютерам і ігровим приставкам не будуть потрібні потужні графічні адаптери (відеокарти), адже вся обробка даних і рендеринг будуть проводитися cloud-серверами, а гравці будуть отримувати вже оброблене відео. Одним із перших заявив про себе сервіс OnLive, і зовсім недавно про це заговорила і компанія Sony, яка збирається впровадити дану ідею в Playstation 3.

Згідно SaaS-концепції користувач платить не одноразово, купуючи продукт, а як би бере його в оренду. Причому, використовує рівно ті функції, які йому потрібні. Наприклад, раз на рік вам потрібна якась програма. І частіше ви її використовувати не збираєтеся. Так навіщо ж купувати продукт, який буде у вас лежати без діла? І навіщо витрачати на нього місце (у квартирі, якщо це коробка з диском, на вінчестері, якщо це файл)?

Конкуренція в хмарній сфері призвела до появи безкоштовних сервісів. Саме по такому шляху пішли два конкуренти – Microsoft і Google. Обидві компанії випустили набори сервісів, що дозволяють працювати з документами. У Google – це Google Docs, у Microsoft – Office Web Apps.

При цьому, обидва сервісу тісно взаємопов'язані з поштою (Gmail в першому випадку і Hotmail у другому) і файловими сховищами. Таким чином, користувача як би переводять зі звичної йому оффлайн-середовища в онлайн. Важливо, що і Google, і Microsoft інтегрують підтримку своїх онлайн-сервісів в усі програмні середовища – як настільні, так і мобільні (нагадаємо, що Google створила ОС Android, а Microsoft – Windows Phone 7).

Аналогічну концепцію (але з дещо іншими акцентами) просуває і головний конкурент обох компаній – Apple. Йдеться про дуже цікавому сервісі під назвою MobileMe. Сервіс включає в себе поштовий клієнт, календар, адресну книгу, файлове сховище, альбом фотографій та інструмент для виявлення загубленого iPhone. За можливість користуватися всім цим Apple бере приблизно 65 євро (або 100 доларів) на рік. При цьому Apple забезпечує такий рівень взаємодії свого набору інтернет-сервісів і додатків на комп'ютері (під управлінням Mac OS X), телефоні, плеєрі і iPad, що необхідність у використанні браузера пропадає. Ви користуєтеся звичними програмами на своєму Mac, iPhone і iPad, однак, всі дані зберігаються не на них, а в хмарі, що дозволяє забути про необхідність синхронізації, а також – про їх доступність.

Якщо Apple інтегрує веб-сервіси у звичні додатки операційної системи, то Google заходить з протилежного боку: операційна система Chrome OS являє собою, фактично, один браузер, через який користувач взаємодіє з розгалуженою мережею веб-сервісів. ОС орієнтована на нетбуки, відзначаються дуже низькі системні вимоги і відсутність необхідності самостійної установки програм (так як всі програми працюють безпосередньо в інтернеті). Тобто Google надає переваги хмарної концепції, зазвичай декламовані при роботі з корпоративними клієнтами, звичайним користувачам. Разом з тим, очевидна неможливість використання таких нетбуків в країнах з недостатньо широким проникненням широкосмугового інтернету. Тому що без інтернету нетбук на базі Chrome OS буде абсолютно даремний.

Всі три типи хмарних сервісів взаємопов'язані, і являють вкладену структуру.

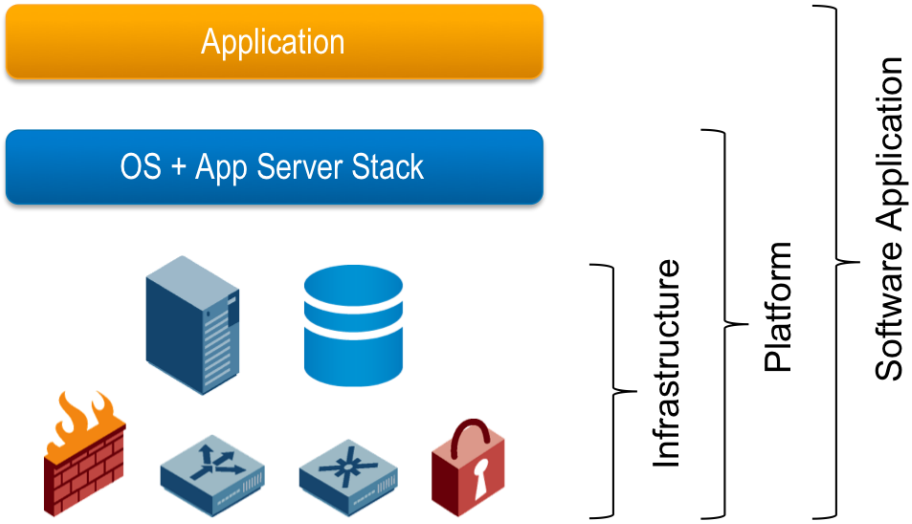


Рисунок 7.5. – Взаємозв'язок хмарних сервісів

Крім яярізних способів надання сервісів розрізняють кілька варіантів розгортання хмарних систем:

Приватна хмара (private cloud) – використовується для надання сервісів всередині однієї компанії, яка є одночасно і замовником і постачальником послуг. Це варіант реалізації «хмарної концепції», коли компанія створює її для себе самої, в рамках організації. У першу чергу реалізація private cloud знімає одне з важливих питань, яке неодмінно виникає у замовників при ознайомленні з цією концепцією – питання про захист даних з точки зору інформаційної безпеки. Оскільки «хмара»обмежена рамками самої компанії, це питання вирішується стандартними існуючими методами. Для private cloud характерне зниження вартості обладнання за рахунок використання простоюють або неефективно використовуваних ресурсів. А також, зниження витрат на закупівлі обладнання за рахунок скорочення логістики (не думаємо, які сервера закуповувати, в яких конфігураціях, які продуктивні потужності, скільки місця кожного разу резервувати і т.д.

По суті, потужність нарощується пропорційно навантаженню, в незалежності від кожної виникаючої задачі, так би мовити, в середньому. І стає легше і планувати, і закуповувати і реалізовувати – запускати нові завдання у виробництво.

Публічна хмара – використовується хмарними провайдерами для надання сервісів зовнішнім замовникам.

Змішана (гібридна) хмара – спільне використання двох перерахованих вище моделей розгортання

Взагалі одна з ключових ідей Cloud полягає саме в тому, щоб з технологічної точки зору різниці між внутрішніми і зовнішніми хмарами не було і замовник міг гнучко переміщати свої завдання між власної та орендованої IT-інфраструктурою, не замислюючись, де конкретно вони виконуються.

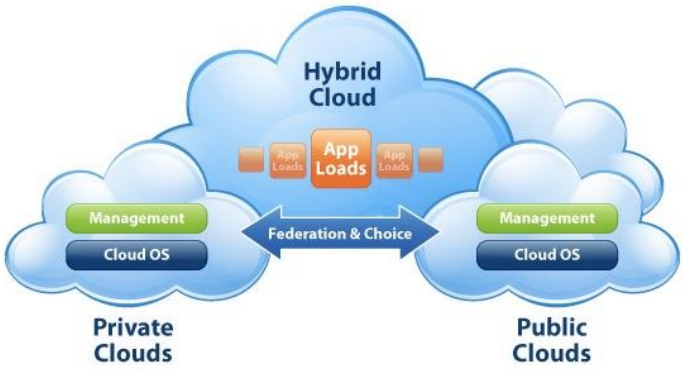


Рисунок 7.6. – Взаємозв'язок хмар різних типів

Таким чином, ці технології при спільному використанні дозволяють користувачам хмарних обчислень скористатися обчислювальними потужностями і сховищами даних, які за допомогою певних технологій віртуалізації і високого рівня абстракції надаються їм як послуги.

7.2. Переваги хмарних обчислень

Розглянемо основні переваги та гідності технологій хмарних обчислень:

Доступність і відмовостійкість-всім користувачам, з будь-якої точки де є Інтернет, з будь-якого комп'ютера, де є браузер.

Клієнтські комп'ютери. Користувачам немає необхідності купувати дорогі комп'ютери, з великим обсягом пам'яті і дисків, щоб використовувати програми через веб-інтерфейс. Також немає необхідності в CD і DVD приводах, так як вся інформація і програми залишаються в «хмарі". Користувачі можуть перейти з звичайних комп'ютерів і ноутбуків на більш компактні і зручні нетбуки.

Доступ до документів. Якщо документи зберігаються в ”хмарі”, вони можуть надаватися користувачам в будь-який час і в будь-якому місці. Більше немає такого поняття як забуті файли: якщо є Інтернет – вони завжди поруч.

Стійкість до втрати даних або крадіжки обладнання. Якщо дані зберігаються в "хмарі", їх копії автоматично розподіляються по декількох серверах, можливо перебувають на різних континентах. При крадіжці або поломці персональних комп'ютерів користувач не втрачає цінну інформацію, яку він до того ж може отримати з будь-якого іншого комп'ютера.

Надійність. Датацентри управляються професійними фахівцями, що забезпечують цілодобову підтримку функціонування віртуальних машин. І навіть якщо фізична машина «завалиться», завдяки розподілу програми на безліч копій воно все одно продовжить свою роботу. Це створює певний високий рівень надійності та відмовостійкості функціонування системи.

Економічність та ефективність – плати стільки, скільки використовуєш, дозволь собі дорогі, потужні комп'ютери і програми. «Хмара»дозволяє враховувати і оплачувати тільки фактично спожиті ресурси строго за фактом їх використання.

Оренда ресурсів. Звичайні сервера середньої компанії завантажені на 10-15 %. В одні періоди часу є потреба в додаткових обчислювальних ресурсах, в інших ці дорогі ресурси простоюють. Використовуючи необхідну кількість обчислювальних ресурсів в «хмарі»в будь-який момент часу, компанії скорочують витрати на обладнання та його обслуговування. Це дає можливість замовнику відмовитися від закупівель дорогих ІТ-активів на користь їх навіть не оренди, а операційного споживання в міру потреби, при скороченні витрат на обслуговування своїх систем та отриманні від постачальника гарантій рівня сервісу.

Оренда ПЗ. Замість придбання пакетів програм для кожного локального користувача, компанії купують потрібні програми в «хмарі». Дані програми будуть використовуватися тільки користувачами, для яких ці програми необхідні в роботі. Більше того, вартість програм, орієнтованих на доступ через Інтернет, значно нижче, ніж їх аналогів для персональних комп'ютерів. Якщо програми використовуються не часто, то їх можна просто орендувати з погодинною оплатою. Витрати на оновлення програм і підтримку в працездатному стані на всіх робочих мріях зовсім зведені до нуля.

Для постачальника ІТ-послуг економічний сенс хмари полягає в ефекті масштабу (обслуговувати великий однорідний центр обробки дешевше, ніж безліч маленьких різнорідних) і згладжування навантаження (коли споживачів багато, малоймовірно, що пікові потужності знадобляться всім їм одночасно).

Розробники ПЗ теж отримують вигоду від переходу в хмари: тепер їм стало простіше, швидше і дешевше розробляти, тестувати під навантаженням і пропонувати клієнтам свої рішення – це можна робити прямо в хмарі з мінімальними витратами. Крім того, Хмарні обчислення – це ефективний інструмент підвищення прибутку і розширення каналів продажів для незалежних виробників програмного забезпечення у формі SaaS. Цей підхід дозволяє організувати динамічне надання послуг, коли користувачі можуть

проводити оплату за фактом і регулювати обсяг своїх ресурсів залежно від реальних потреб без довгострокових зобов'язань.

Простота – не потрібна покупка і налаштування програм та обладнання, їх оновлення.Обслуговування. Так як фізичних серверів з впровадженням Cloud Computing стає менше, їх стає легше і швидше обслуговувати. Що стосується програмного забезпечення, то останнє встановлено, налаштовано і оновлюється в «хмарі». У будь-який час, коли користувач запускає віддалену програму, він може бути впевнений, що ця програма має останню версію – без необхідності щось встановлювати заново або платити за оновлення.

Спільна робота. При роботі з документами в «хмарі»немає необхідності пересилати один одному їх версії або послідовно редагувати їх. Тепер користувачі можуть бути впевненими, що перед ними остання версія документа і будь-яка зміна, внесена одним користувачем, миттєво відбивається в іншого.

Відкриті інтерфейси. «Хмара» як правило, має стандартні відкриті API (інтерфейси прикладного програмування) для зв'язку з існуючими додатками і розробки нових – спеціально для хмарної архітектури.

Гнучкість і масштабованість – необмеженість обчислювальних ресурсів (пам'ять, процесор, диски). «Хмара» масштабованість і еластично – ресурси виділяються і звільняються у міру потреби.

Продуктивні обчислення. У порівнянні з персональним комп'ютером обчислювальна потужність, доступна користувачу «хмарних»комп'ютерів, практично обмежена лише розміром «хмари», тобто загальною кількістю видалених серверів. Користувачі можуть запускати більш складні завдання, з великою кількістю необхідної пам'яті, місця для зберігання даних, тоді, коли це необхідно. Іншими словами, користувачі можуть за бажання легко і дешево попрацювати з суперкомп'ютером без будь-яких фактичних придбань. Можливість запуску безліч копій програми на багатьох віртуальних машинах представляє переваги масштабованості: кількість примірників програми здатне практично миттєво збільшуватися на вимогу, залежно від навантажень.

Зберігання даних. У порівнянні з доступним місцем для зберігання інформації на персональних комп'ютерах обсяг сховища в «хмарі» може гнучко і автоматично підлаштовуватися під потреби користувача. При зберіганні інформації в «хмарі» користувачі можуть забути про обмеження, що накладаються звичайними дисками, – «хмарні» розміри обчислюються мільярдами гігабайт доступного місця.

Інструмент для стартапів. В очах таких споживачів сервісу хмарних обчислень як компанії, початківці свій бізнес основною перевагою даної технології є, відсутність необхідності закуповувати всі відповідне обладнання і ПЗ, а потім підтримувати їх роботу.

7.3. Недоліки та проблеми хмарних обчислень

Чи є мінуси в «хмарних» обчислень? Чому «хмарні» технології в Росії тільки набирають обертів, а директори деяких великих компаній не поспішають

переводити IT-інфраструктуру своїх підприємств в «хмари»? Отже, відзначимо основні недоліки і труднощі використання cloud computing:

Постійне з'єднання з мережею. Cloud Computing завжди майже завжди вимагає з'єднання з мережею (Інтернет). Якщо немає доступу в мережу – немає роботи, програм, документів. Багато «хмарні» програми вимагають хорошого Інтернет-з'єднання з великою пропускною здатністю. Відповідно програми можуть працювати повільніше ніж на локальному комп'ютері. На думку провідних російських IT-компаній, основною перешкодою широкому розвитку хмар, є відсутність широкосмугового доступу в Інтернет (ШСД) – насамперед у регіонах.

7.4. Безпека

Безпека даних теоретично може бути під загрозою. Не всі дані можна довірити сторонньому провайдеру в інтернеті, тим більше, не тільки для зберігання, але ще й для обробки. Все залежить від того, хто надає «хмарні» послуги. Якщо цей хтось надійно шифрує Ваші дані, постійно робить їх резервні копії, вже не один рік працює на ринку подібних послуг і має хорошу репутацію, то загрози безпеки даних може ніколи не статися. У користувача «хмарних» бізнес додатків можуть також виникнути і юридичні проблеми, наприклад пов'язані з виконанням вимог захисту персональних даних.

Держава, на території якого розміщено датацентр, може отримати доступ до будь-якої інформації, яка в ньому зберігається. Наприклад, за законами США, де знаходиться найбільша кількість датацентрів, в цьому випадку компанія-провайдер навіть не має права розголошувати факт передачі конфіденційної інформації будь-кому, крім своїх адвокатів.

Ця проблема є, напевно, однією з найбільш суттєвих в питанні виведення конфіденційної інформації в хмару. Шляхів її вирішення може бути декілька. По-перше, можна шифрувати всю інформацію, що поміщається на хмару. По-друге, можна просто її туди не поміщати. Однак, у всякому разі, у компаній, що користуються хмарними обчисленнями, це має бути визначеним пунктом в списку питань інформаційної безпеки. Крім того, самі провайдери повинні покращувати свої технології, надаючи деякі послуги з шифрування.

Функціональність «хмарних» додатків. Не всі програми або їх властивості доступні віддалено. Якщо порівнювати програми для локального використання і їх «хмарні» аналоги, останні поки програють у функціональності. Наприклад, таблиці Google Docs або програми Office web application мають набагато менше функцій і можливостей, ніж Microsoft Excel.

7.5. Залежність від «хмарного» провайдера

Завжди залишається ризик, що провайдер онлайнових сервісів одного разу не зробить резервну копію даних – якраз перед падінням сервера. Ризик цей, втім, навряд чи перевищує небезпеку того, що користувач сам упустить свої дані – втративши або розбивши мобільник або ноутбук, не створивши на

домашньому ПК резервну копію. Крім того, прив'язавшись до тієї чи іншій послузі, ми якоюсь мірою також обмежуємо свою свободу – свободу переходу на стару версію софтвера, вибору способів обробки інформації і так далі.

Деякі експерти, наприклад Г. Маклеод (Hugh Macleod) у статті «Найбільш добре охороняється секрет Хмар», стверджують, що хмарні обчислення ведуть до створення величезної, небаченої раніше монополії. Чи можливо це? Звичайно, на ринку хмарних обчислень для приміщення в хмару якої інформації, у відношенні якої існують правила інформаційної безпеки, компанії будуть скоріше використовувати таких вендорів, чиє ім'я «на слуху» і кому вони довіряють. Таким чином, існує певна небезпека того, що всі обчислення і дані будуть агреговані в руках однієї сверхмонополіі. Проте на даний момент на ринку вже існують кілька компаній з приблизно однаковим високим рівнем довіри з боку клієнтів (Microsoft, Google, Amazon), і немає ніяких фактів, які б вказували на можливість домінування однієї підприємством всіх інших. Тому в найближчому майбутньому поява глобальної сверхкомпанії, яка буде координувати і контролювати всі обчислення в світі, дуже малоймовірно, хоча одна лише можливість такої події відлякує деяких клієнтів.

7.6. Перешкоди розвитку хмарних технологій

Недостатня довіра споживачів хмарних послуг. Нерідко бізнес належить до хмарним послуг декілька насторожено. «Причин же недовірливого ставлення малого та середнього бізнесу до хмарним дата-центрам може бути декілька. Швидше за все, це боязнь втратити контроль над IT-ресурсами, побоювання щодо гарантії збереження і захисту переданої інформації і представлення дата-центру лише як майданчика для розміщення обладнання».

Канали зв'язку в більшості регіонів країни характеризуються відсутністю SLA за якістю наданого сервісу (QoS), що особливо відноситься до останніх милям. Що толку від того, що ваш основний трафік йде по магістралі з гарантованим QoS (зі своїми обмеженнями), якщо кінцеві умови підключені до неї через місцевого оператора, навіть не чула про таку проблему. При цьому вартість зв'язку для великих організацій може складати до 50 % від IT-бюджету. Відповідно перехід до хмарної моделі істотно впливає на мережеву топологію ваших потоків даних і, швидше за все, QoS буде гірше ніж у внутрішній мережі. Або що б отримати якість обслуговування на прийнятному рівні доведеться заплатити стільки грошей, що вся економія від централізації інфраструктури або додатків буде перекреслена зростанням комунікаційних витрат.

Безпека. Проблема безпеки є серйозним стримуючим фактором. Нерідко Служби Безпеки створюють досить високий загороджувальний бар'єр для ідеї винести будь-які дані за периметр своєї мережі. Часто без якої нормальної аргументації.

Відсутність надійних ЦОДів. З приводу центрів обробки даних (ЦОД) досить згадати, що в країні, здається, немає ще жодного Tier III ЦОДа за класифікацією Uptime Institute. Цілком зрозуміло, що їх поява – це питання

часу. Через кризу більшість будівництв було заморожено або відкладено. Тим не менш, поки достатньої інфраструктури в країні просто немає.

7.7. Розподілені обчислення (grid computing)

Відзначимо на закінчення ще одну технологію, яка з одного боку також зробила вплив на появу концепції хмарних обчислень, а з іншого боку має ряд істотних відмінностей. Йдеться про колективні, або розподілені обчислення (grid computing) – коли велика ресурсномістка обчислювальна задача розподіляється для виконання між безліччю комп'ютерів, об'єднаних в потужний обчислювальний кластер мережею в загальному випадку або інтернетом зокрема.

Встановлення загального протоколу в мережі Інтернет безпосередньо призвело до швидкого зростання онлайн користувачів. Це призвело до необхідності виконувати більше змін в поточних протоколах і до створення нових. На поточний момент широко використовується протокол Ipv4 (четверта версія IP протоколу), але обмеження адресного простору, заданого ipv4, неминуче призведе до використання протоколу ipv6. Протягом довгого часу удосконалилося апаратне і програмне забезпечення, в результаті чого вдалося побудувати загальний інтерфейс в Інтернет. Використання веб-браузерів призвело до використання моделі «Хмари», замість традиційної моделі інформаційного центру.

На початку 1990-их, Іен Фостер і Карл Кесселмен представили їх поняття Грід обчислень. Вони використовували аналогію з електричною мережею, де користувачі могли підключатися і використовувати послугу. Грід обчислення в чомусь спираються на методах, що використовуються в кластерних обчислювальних моделях, де багаторазові незалежні групи, діють, як мережа просто тому, що вони не всі розташовані в межах тієї ж області.

Зокрема, розвиток Грід технологій дозволило створити так звані GRID – мережі, в яких група учасників могла спільними зусиллями вирішувати складні завдання. Так, співробітники IBM створили інтернаціональну команду grid – обчислень, що дозволила істотно просунутися в області боротьби з вірусом імунного дефіциту. Цілі команди з різних країн приєднували свої обчислювальні потужності і допомогли «обрахувати» і змоделювати найбільш перспективні форми для створення ліків від СНІДу...»

На практиці межі між цими (grid і cloud) типами обчислень досить розмиті. Сьогодні з успіхом можна зустріти «хмарні» системи на базі моделі розподілених обчислень, і навпаки. Однак майбутнє хмарних обчислень все ж значно масштабніше розподілених систем, до того ж не кожен «хмарний сервіс» вимагає великих обчислювальних потужностей з єдиної керуючої інфраструктурою або централізованим пунктом обробки платежів.

Короткі підсумки:

Ми розглянули основні поняття хмарних обчислень, приклади, особливості, основні різновиди хмарних технологій, їх переваги і недоліки.

Ключові терміни:

Хмарні обчислення – технологія обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс.

Інфраструктура як сервіс – це надання комп'ютерної інфраструктури як послуги на основі концепції хмарних обчислень.

Платформа як сервіс – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб-додатків як послуги.

Програмне забезпечення як сервіс – модель розгортання програми, яка має на увазі надання додатка кінцевому користувачеві як послуги на вимогу. Доступ до такого додатку здійснюється за допомогою мережі, а найчастіше за допомогою Інтернет-браузера.

Приватна хмара – це варіант локальної реалізації «хмарної концепції», коли компанія створює її для себе самої, в рамках однієї організації.

Публічне хмара – використовується хмарними провайдерами для надання сервісів зовнішнім замовникам.

Розподілені обчислення – технологія коли більша ресурсномістка обчислювальна задача розподіляється для виконання між безліччю комп'ютерів, об'єднаних в потужний обчислювальний кластер мережею або інтернетом.

Литература:

1. John W. Rittinghouse, James F. Ransome – «Cloud Computing: Implementation, Management, and Security»
2. Tim O'Reilly «Web 2.0 and Cloud Computing» radar.oreilly.com/2008/10/web-20-and-cloud-computing.html
3. Г. Маклеод (Hugh Macleod) «Самый хорошо охраняемый секрет Облаков»technorati.com/posts/lv3vwaZ9hbuGSZx_jQseIqaVSlj29LQGjWyRkNoZ4b0%3D?reactions
4. cloudcomputingexpo.com/
5. http://habrahabr.ru/blogs/Azure/60100/

ЛЕКЦІЯ 8. МОДЕЛІ ІНФРАСТРУКТУРИ «ХМАРНИХ» ОБЧИСЛЕНЬ. КОНСОЛІДАЦІЯ ДАНИХ

Сучасний стан розвитку інформаційного суспільства характеризується поєднанням глобальних інформаційно-комунікаційних систем, інтеграцією баз даних та знань, зростанням спектру послуг та сервісів, а також різким збільшенням клієнтського навантаження на серверний простір. Таким чином формується чітка не відповідність між бурхливим зростанням обсягів інформаційних потоків і мережних сервісів та сучасним технічним станом програмно-апаратного забезпечення мережевої інфраструктури інформаційних систем. Зазначене протиріччя можливо вирішити на основі сучасної організації та впровадженню великого класу хмарних технологій, які поступово стають невід'ємною частиною IT інфраструктури суспільства та держави.

З появою більш функціонального програмного забезпечення, ефективних апаратних та інформаційно-комунікаційних технологій, термін «хмарні» технології отримав нове особливе значення і відкрив простір для широкого класу базових і спеціальних сервісів інформаційних систем. Основним завданням впровадження хмарних технологій є надання користувачам якісних послуг з гарантованою якістю в умовах збереження основних властивостей інформаційних ресурсів: цілісності, доступності конфіденційності.

8.1. Порівняльний аналіз моделей хмарних технологій

Хмарні обчислення (англ. Cloud computing) – це новітня технологія, яка забезпечує повсюдний і зручний мережевий доступ до загального пулу обчислювальних ресурсів, які можуть бути надані без значних апаратних затрат та звернення до провайдера або власника ресурсів [1].

До обов'язкових сучасних характеристик і можливостей хмарних технологій, відносяться:

– самообслуговування клієнта при роботі в «хмарах» на основі надання провайдером (або власником інформаційних ресурсів) при умовах самостійного вибору користувачем переліку послуг, задач, обчислювальних потреб та інших встановлених сервісів системи;

– організація системи універсального віддаленого доступу за системою встановлених ідентифікаторів до інформаційних ресурсів мережі при умові використання послуг незалежно від основного терміналу провайдера (або власника ресурсів);

– системне управління провайдером (або власником ресурсів) апаратно-програмними, інформаційними потужностями шляхом їх інтеграції та динамічного перерозподілу між клієнтами;

– система автоматичної організації роботи користувачем з провайдером послуг при умовах еластичної зміни наданих сервісів згідно вимог клієнтів;

– організація провайдером системи автоматичного обліку та аудиту використання інформаційних ресурсів системи;

– захист інформаційних ресурсів системи та безпосередньо їх властивостей: цілісності, доступності, конфіденційності.

Базовою рисою впровадження хмарних технологій, з точки зору бізнес процесів, є допомога провайдера послуг зменшити інформаційно-технічне навантаження на програмно-апаратні комплекси замовника та в значній мірі реалізувати політику фінансово-технічної економії задіяних масштабів інформаційних ресурсів. У свою чергу, автоматизація процедур модифікації інформаційної системи власником ресурсів, зменшує обсяги витрат на обслуговування більшої кількості абонентів.

8.2. Відмовостійкість та масштабованість системи

Використання хмарних технологій має забезпечувати цілковиту та постійну доступність всіх клієнтів послуг до ресурсів. Однак на практиці, компоненти інфраструктури мають обмежений технічний ресурс та зовнішні навмисні і не навмисні впливи, які приводять до часткового чи повного відказу системи (підсистем), а значить втрати доступності ресурсів. Результатом реалізації таких загроз є відмова у роботі та наданні сервісів дата центрами. Тому при впровадженні хмарних технологій необхідно розробляти та впроваджувати складні забезпечувальні процеси їх функціонування у складних умовах. На основі зазначеного, вкрай необхідна організація таких додаткових процесів в роботі системи, як: контроль робочих характеристик, аудит процесів і розслідування інцидентів, гаряче резервування програмно-апаратних комплексів і каналів зв’язку, побудова системи захисту інформаційних ресурсів, тощо. Прикладом таких процедур може бути перезапис додатків з гарячим резервуванням.

Хмарні технології повинні мати одну з головних рис ефективної роботи в сучасних умовах інтегрованого інформаційного середовища: відмовостійкість. Розробляючи відмовостійкі системи, слід звернути увагу на внутрішню архітектуру додатків, враховуючи можливість їх розподіленого виконання. Додаток повинен мати модульну структуру, при цьому модулі підтримують зв'язок між собою за допомогою стандартизованих інтерфейсів програмування. У випадку збою, така ідеологія побудови структури забезпечить можливість відновлення роботи програми значно швидше, ніж у випадку відсутності модулів. Використання технологій, що дозволяють підтримувати хоча б низький рівень роботи додатку, у випадку, коли його модулі знаходяться на відключених серверах, забезпечить безперервну роботу користувача послуг при значних несправностях серверної частини або часткової її відмови.

Другою з значних переваг використання хмарних технологій є її масштабованість. Під масштабованістю інформаційної системи слід розуміти можливість адаптації, динамічного перерозподілу процесів, сервісних додатків, послуг та інформаційних ресурсів системи в умовах розвитку та масштабного

розростання загальної інфраструктури системи. Використання методу безпосереднього збільшення фізичних ресурсів, може привести до тупикової ситуації, коли просте нарощування технічних ресурсів більш не можливе. Властивість динамічного масштабування необхідно закладати ще на етапах проектування системи.

8.3. Моделі та технології організації

Сучасні інформаційні технології використовують чотири моделі розгортання хмарних сервісів:

– Суспільна хмара (англ. communitycloud) – вид інфраструктури, призначений для використання певною групою користувачів, що об'єднані спільним класом інформаційних або технічних інтересів, інформаційних потоків даних, мають спільні задачі, тощо.

– Гібридна хмара (англ. hybridcloud) – це поєднання декількох хмарних інфраструктур, що залишаються унікальними інформаційними об'єктами, але пов'язані між собою приватними або стандартизованими технологіями передачі даних.

– Публічна хмара (англ. publiccloud) – це інфраструктура доступна для широкої громадськості і знаходиться у власності організації продажу хмарних сервісів та послуг системи.

– Приватна хмара (англ. privatecloud) – це інфраструктура, призначена для використання однією організацією. Приватна хмарна це інфраструктура використовується виключно для вирішення задач стандартизованого класу послуг і технічних можливостей організації. Зазначена інфраструктура може підлягати управлінню, як з сторони безпосередньо самої організації, так і з сторони третіх осіб. Однією з переваг організації приватної хмари є то, що вона може існувати, як всередині так і поза територіальними повноваженнями організації.

Сьогодні існує багато типів хмарних технологій, однак ми розглянемо три основні моделі технологій, а саме IAAS, PAAS та SAAS.

Infrastructure as a Service (IAAS) – інфраструктура як сервіс. Провайдер послуг пропонує для оренди фізичні або віртуальні (в більшості випадків) інформаційні ресурси в основному для організації і розміщення центрів обробки даних (ЦОД) та їх серверів.

Прикладом використання такої моделі може бути Amazon Cloud Drive, Windows Azure, Rackspace, SkyDrive та інші представлені в таблиці 8.1 приклади ПЗ.

Таблиця 8.1.

Порівняння персональних хмарних сховищ

	AmazonCloud Drive	**Dropbox**	**SkyDrive**	**iCloud**	**Google Drive**	**WUALA**
Безкоштовне зберігання (ГБ)	5	2	7	5	15	5
Вартість за додаткові 100 ГБ	50 $/рік	99 $/рік	51 $/рік	112 $/рік	60 $/рік	119.88 $/рік
Максимальна ємність	1000 ГБ	500 ГБ	107 ГБ	55 ГБ	16 ТБ	2 ТБ
Вартість за ГБ	$0.50	$1.00	$0.51	$2.24	$0.60	$0.59
Програмне забезпечення он-лайн	Хмарний плеєр (тільки музика)	PDF and Picture Viewer	Редактор для docx, xlsx, pptxта picture viewer	Редактор для.doc, xls, and.ppt	Редактор для.doc, xls, ppt, diagrams та нотаток.	Редактор для doc, xls, and.ppt
Настільний клієнт синхронізації	Да	Да	Да	Да	Да	Да

Software as a Service (SAAS) – програмне забезпечення, як сервіс. Основна та найпопулярніша послуга, що надається в рамках хмарних технологій, що забезпечує доступ та використання програмних продуктів без його встановлення та налаштування на робочий машині клієнта. До основних переваг SAAS систем можна віднести технічну сторону, підтримкою і обслуговуванням якої займається компанія, що надає послуги та контроль процесів – можливість отримання віддаленого доступу без територіальних обмежень.

Прикладом моделі обслуговування SAAS є GoogleApps, SalesForce.Com, Webex, Office 365; для пошти користувачів – Gmail, Hotmail; для користувацьких зображень – Flickr, Picasa. Порівняльна характеристика найвідоміших сервісів представлена в таблиці 8.2.

Таблиця 8.2.

Порівняння найвідоміших моделей SAAS

	Office 365	**GoogleApps**
Вартість, $/користувача/міс	5	5,10
Почтовий сервіс, ГБ	Exchange, 50	Gmail, 30
Програмне забезпечення он-лайн	Документи, Таблиці, Презентації	docx, .xlsx, .pptx
Робота з мобільних приладів	Середня	Відмінна
Сховище документів	SkyDrivePro	GoogleDisk
Розмір сховища, ГБ	7	20
Відеоконференія	до 250 учасників	до 15 учасників
«Хмарний» зовнішній доступ до документів і спільна робота з ними	SharePoint Online дозволяє створювати повнофункціональні програми для спільної роботи, вимагає попереднього налаштування	GoogleGroups, не вимагає попереднього налаштування та більш простий у використанні

Крім того необхідно зазначити, що модель SAAS переважно використовується для організацій, які працюють в одній галузі і відрізняються визначеними класами інформаційних потоків і даних, або мають у своєму складі багато територіально-розподілених структурних одиниць. До таких організацій відносяться міністерства або відомства, такі як Міністерство освіти і науки Украъни, Міністерство внутрішніх справ України, великі навчальні заклади, такі як Національній авіаційний університет, Національний технічний університет України "Київський політехнічний інститут" та інші.

Platform as a Service (PAAS) – платформа, яка дозволяє розробникам додатків розміщати свої прикладні програми, розроблені за допомогою мов програмування, бібліотек, сервісів та інструментів наданих хмарним провайдером при умові того, що користувач сервісу не має можливості контролювати інфраструктуру хмари, але може контролювати розроблені додатки. Прикладом є AmazonWebServices, Heroku, Force.com, GoogleAppEngine, найвідоміші представлені в таблиці 8.3.

Таблиця 8.3.

Порівняння найвідоміших моделей PAAS

	GoogleAppEngine	**AmazonWebServices**
Ціна за дисковий простір	$0.15 за Гб/міс	$0.08-$0.15 за Гб/міс
Вихідний трафік	$0.12/Гб	$0.08-$0.15/Гб
Підтримка Java	Так	Так
Легкість масштабування	Так	Так
Гнучкість	Обмежена	Так Так
Безкоштовний процесорний час	6.5 CPU годин за добу	-
Безкоштовний дисковий простір	1 Гб	-

8.4. Організація безпеки хмарних технологій

Впровадження хмарних технологій має ряд переваг, однак все частіше постає питання організації комплексної системи захисту інформаційних ресурсів хмари, як для серверної, так і для клієнтської частини.

Для надійної та безпечної роботи додатків в глобальному інформаційному просторі потрібно дотримуватись певних рекомендацій, згідно міжнародних та вітчизняних стандартів.

Корпоративні додатки мають забезпечувати ідентифікацію особи користувача, аутентифікацію та розподілення прав доступу до системи і її ресурсів. Корпоративна безпека може забезпечуватися, як спеціальними сервісами, так і вбудованими в продукти рішеннями [2-4].

Комплексна система захисту ресурсів системи та її додатків, повинна закладатися в хмарну технологію, ще на етапах проектування і впровадження. Рекомендовано мати багаторівневу систему безпеки, при цьому захист хмари і додатків повинні бути взаємопов'язані. Методи шифрування слід використовувати як для передачі, так і для зберігання даних.

Для забезпечення ефективності захисту ресурсів потрібно проводити тестування додатків в умовах, наближених до реальних подій. Однак, досі залишається відкритим питання захищеності каналу зв’язку при організації передачі даних з використанням хмарних технологій.

Висновок

Проведено порівняльний аналізу сучасних моделей побудови, обслуговування та сервісу хмарних технологій. Визначено сучасні технічні характеристики, проведено класифікацію та встановити переваги різних типів хмарних технологій. Досліджено моделі розгортання хмарних сервісів. Визначно критерії порівняння найвідоміших програмних моделей IAAS, PAAS та SAAS.

Література

1. Mell P., Grance T. The NIST Definition of Cloud Computing (Draft) // Recommendations of the National Institute of Standardsand Technology. Special Publication 800-145 (Draft). 2011. P. 1 – 3.

2. Юдін О. К. Безпека інформаційно-комунікаційних систем. питання термінології та базових визначень галузі // Наукоємні технології. – 2012. – Т. 14. – №. 2. – С. 86 – 91.

3. Юдін О.К. Інформаційна безпека. Нормативно-правове забезпечення/ О.К. Юдін. – К.: Вид-во Нац. авіац. ун-ту «НАУ-друк», 2011. – 640 с.

4. Юдін О.К., Корченко О.Г., Конахович В.Г. Захист інформації в мережах передачі даних // К.: Вид-во ТОВ „НВП „Інтерсервіс. – 2009.

ЛЕКЦІЯ 9. ХМАРИ ГЕТЕРОГЕННИХ РЕСУРСІВ. «ХМАРНІ» ОБЧИСЛЕННЯ ТА GRID-КОМП'ЮТІНГ. WEB-СЛУЖБИ В ХМАРІ

Розглянемо деякі з веб-служб, що надаються концепцією хмарних обчислень. Інфраструктура є послугою в концепції хмарних обчислень. Є багато різновидів управління інфраструктурою в хмарному середовиші. «Інфраструктура як Сервіс» (Infrastructure – as-a-Service, IaaS) в основному за запитом на базі сучасних обчислювальних технологій і високошвидкісних мереж. «Комунікацій як Сервіс»(Communication-as-a-Service, CaaS).»Програмне забезпечення як Сервіс «(Software-as-a-Service, SaaS), такі як Amazon.com з їх еластичною платформою хмари, характеристики, переваги, та архітектурний рівень обслуговування. Досліджуємо ключові особливості використання зовнішніх джерел/ресурсів (outsourcing), доступні як «Платформи як Сервіс»(Platforms-as-a-Service, PaaS).

Оскільки технології мігрують від традиційної локальної моделі в нову модель хмари, сервісні пропозиції розвиваються практично щодня. Пропозиції веб-служб часто мають багато спільних характеристик. Часто від клієнта потрібні лише мінімальні витрати для отримання послуги. Масштабованість передбачається для кожного з типів пропозицій, але це незавжди необхідно. Багато хто з «хмарних» вендорів ще працюють над використанням масштабованості, тому що їх користувачі поки не потребують даному виді послуг. Нарешті, пристрій і незалежність місця розташування дозволяє користувачам отримати доступ до систем незалежно від того, де вони знаходяться або які пристрої використовують.

Інфраструктура як Сервіс (Infrastructure-as-a-Service, IaaS) – надання комп'ютерної інфраструктури (як правило, це платформи віртуалізації) як сервісу. IaaS посилює технологію, послуги і вкладення в центри обробки даних, щоб надати це як послугу клієнтам. На відміну від традиційного аутсорсингу, який вимагає належної старанності, нескінченних переговорів і складних, довгих контрактів, IaaS зосереджена навколо моделі надання послуг, яка забезпечує зумовлену, стандартизовану інфраструктуру, безумовно оптимізовану під потреби клієнта. Спрощені пропозиції роботи і вибір рівня сервісного обслуговування полегшує клієнтові вибір рішення з певним набором основних експлуатаційних характеристик. Як правило, постачальники надають компоненти наступних рівнів:

– Апаратне забезпечення (як правило, Грід з масивною горизонтальною масштабованістю).

– Комп'ютерна мережа (включаючи маршрутизатори, брандмауери, балансування навантаження і т.д.).

– Підключення Інтернет.

– Платформа віртуалізації для того, щоб запускати віртуальні машини.

– Угоди сервісного обслуговування.

– Інструменти обліку обчислень.

Замість покупки простору в центрах обробки даних, серверів, програмного забезпечення, мережевого обладнання, і т.д., клієнти IaaS по суті орендують, які знаходяться на стороні обслуговуючих постачальників послуг IaaS. Оплата за надання послуг зазвичай проводиться щомісяця. Клієнт платить тільки за спожиті ресурси. Основні переваги даного типу послуг включає:

– Вільний доступ до попередньо сконфігурованого середовища.

– Використання інфраструктури останнього покоління.

– Захищені і ізольовані обчислювальні платформи.

– Зменшення ризику, використовуючи сторонні ресурси, підтримувані третіми особами.

– Здатність керувати піковими навантаженнями.

– Більш низькі витрати.

– Менший час, вартість і складність при додаванні або розширенні функціональності.

Обчислення на вимогу набувають все більшої популярності серед підприємств. Обчислювальні ресурси, які обслуговують користувача веб сайти, стають все менше і менше, в той час, як доступні ресурси постачальників послуг постійно зростають. Модель на вимогу розвинулася, щоб подолати проблему того, як ефективно задовольнити вимогам системи до ресурсів. Попит на обчислювальні ресурси може істотно змінюватися за досить короткі проміжки часу, і підтримка ресурсів достатніх, щоб задовольнити піковим вимогам може бути дорогою. Технічно складність рішення може бути настільки ж несприятливим, як ситуація, коли підприємство скорочує витрати, підтримуючи тільки мінімальні обчислювальні ресурси.

Рисунок 9.1. – Грід обчислення

Такі поняття як кластерні обчислення, Грід обчислення і т.д., можуть здаватися дуже подібними поняттю обчислень на вимогу, але краще їх зрозуміти можна, якщо думати про них, як стандартних блоках, які розвивалися протягом довгого часу, щоб досягти сучасної моделі хмарних обчислень, яку ми використовуємо сьогодні.

9.1. Amazon

Розглянемо один з прикладів–Amazon's Elastic Compute Cloud (Amazon EC2). Amazon EC2 – веб-служба, яка забезпечує обчислювальні потужності порядкового розміру в хмарі. Це розроблено, щоб зробити веб-обчислення доступнішими для розробників і щоб запропонувати безліч переваг для клієнтів:

– Інтерфейс веб-служби, дозволяє клієнтам отримувати і формувати простір з мінімальним зусиллям.

– Надає користувачам повний контроль над їх (орендованими) обчислювальними ресурсами і дозволяють їм працювати в перевіреному обчислювальному середовищі.

– Зменшує час, необхідний для отримання та завантаження нового сервера до хвилин, дозволяючи клієнтам швидко змінювати конфігурацію, згідно їх обчислювальним вимогам.

– Змінює економіку обчислень, дозволяючи клієнтам платити тільки за використовувані ресурси.

– Надає розробникам інструменти, яких необхідні для побудови відмовостійких додатків і ізолювання себе від загальних сценаріїв відмови.

Amazon EC2 представляє обчислювальне навколишнє середовище, дозволяючи клієнтам використовувати веб інтерфейс, для отримання та управління послугами, необхідними для запуску одного або більше примірників операційної системи. Клієнти можуть завантажити навколишнє середовище з їх налаштованими додатками. Вони можуть керувати мережевими правами доступу і так багато систем, скільки їм потрібно. Для використання Amazon EC2, клієнтам спочатку необхідно створити Amazon Machine Image (AMI). Цей образ містить додатки, бібліотеки, дані та пов'язані параметри конфігурації, використовувані в віртуальному обчислювальному середовищі. Amazon EC2 пропонує використання попередньо сконфігуровані образи, створені з шаблонами, необхідними для негайного запуску. Коду користувачі визначили і формували їх AMI, вони використовують інструменти Amazon EC2 для завантаження образу в Amazon S3. Amazon S3 – склад, який забезпечує безпечний, надійний і швидкий доступ до клієнта AMI. Перш, ніж клієнти зможуть використовувати AMI, вони повинні використовувати веб інтерфейс Amazon EC2 для налаштування безпеки і мережевий доступ.

Під час конфігурації користувачі вибирають, який тип категорії і яку операційну систему вони хочуть використовувати. Доступні типи категорій складають дві різні категорії: Стандартний процесор або High-CPU процесор. Більшості додатків найкраще задовольняє стандартний випадок, в який входять маленький, великий і дуже великий примірники платформи. У разі High-CPU використовується пропорційно більше ресурсів центрального процесора, ніж пам'яті довільного доступу (RAM), що більш підходить високонавантажених додаткам. У разі High-CPU процесора для вибору є середня і дуже велика платформи. Після визначення, яку сутність використовувати, клієнти можуть запустити, завершити, і контролювати так багато примірників їх AMI, скільки

необхідно при використанні інтерфейсів прикладного програмування веб-служби (API) або велику різноманітність інших інструментів управління, якими здійснюється обслуговування. Користувачі можуть вибрати, чи хочуть вони запускати додатки в різних центрах обробки даних, використовувати статичні IP адреси кінцевих точок, при цьому вони платять тільки за фактично споживані ресурси. Користувачі також можуть вибрати доступні AMI з бібліотеки. Наприклад, якщо необхідний звичайний Linux сервер, то клієнтом може бути один із стандартних Linux збірок AMIs.

Є досить багато особливостей сервісу EC2, які забезпечують суттєві переваги для підприємства. Насамперед, Amazon EC2 забезпечує фінансову вигоду. Через великий масштаб компанії Amazon і великої клієнтської бази, це недорога альтернатива багатьом іншим можливим рішенням. Витрати понесені на запуск і керування розділені між великою кількістю клієнтів, роблять вартість для будь-якого клієнта набагато нижче, ніж будь-яка інша альтернатива. Клієнти платять дуже низький відсоток за обчислювальні потужності, які вони фактично споживають. Безпека також забезпечена через інтерфейси веб-служби Amazon EC2. Ці інтерфейси дозволяють користувачам формувати параметри налаштування брандмауера, які контролюють мережевий доступ до і між групами примірників служб. Amazon EC2 пропонує дуже надійне середовище, де випадки заміни можуть бути швидко забезпечені.

– Динамічна Масштабованість.

Amazon EC2 дозволяє користувачам збільшувати або зменшувати продуктивність за кілька хвилин. Користувачі можуть запускати єдиний екземпляр, сотні примірників або навіть тисячі екземплярів служб одночасно. Усім цим керують за допомогою API веб-служби, додаток може автоматично масштабувати себе вгору або вниз, залежно від його потреб. Даний тип динамічної масштабованості дуже привабливий для клієнтів підприємств, тому що це дозволяє їм відповідати вимогам своїх клієнтів, не маючи необхідність добудовувати їх інфраструктуру.

– Повний контроль над екземплярами.

У користувачів є повний контроль над їх екземплярами. У них є повний доступ до кожного примірника, і вони можуть взаємодіяти з ними з будь-якої машини. Примірники можуть бути перезавантажені, віддалено використовуючи API веб-служби. Користувачі також мають доступ до консолі своїх примірників. Як тільки користувачі налаштували їх аккаунт і завантажили їх AMI в сервіс Amazon S3, їм необхідно тільки запустити екземпляр. Можливо запустити AMI на будь-якому числі примірників (або для будь-якого типу), викликавши RunInstances API, який підтримується Amazon.

– Гнучкість конфігурації.

Параметри налаштування конфігурації можуть значно відрізнятися серед користувачів. У них є вибір з різних типів екземплярів, операційних систем, і пакетів програмного забезпечення. Amazon EC2 дозволяє їм вибирати конфігурацію пам'яті, центрального процесора, і системи зберігання, яка оптимально підходить для їх вибору операційної системи та програми. Наприклад, вибір користувача операційної системи може також включати

численні збірки Linux, Microsoft Windows Server, і навіть OpenSolaris, всі запущені на дійсних серверах.

– Інтеграція з Іншими Веб-службами Amazon.

Amazon EC2 працює в з'єднанні з безліччю інших веб – служб Amazon. Наприклад, Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB, Amazon Simple Queue Service (Amazon SQS) і Amazon CloudFront всі інтегровані, щоб забезпечити повне рішення для обчислень, обробки запитів та зберігання між широким діапазоном додатків.

Amazon S3 забезпечує інтерфейс веб-служб, який дозволяє користувачам зберігати і відновлювати будь-який обсяг даних через інтернет в будь-який час, будь-де. Це надає розробникам прямий доступ до того ж самого, добре масштабованості, надійному, швидкому, недорогому використання інфраструктури зберігання даних Amazon, щоб управляти їх власної глобальної мережею веб сайтів. Служба S3 прагне максимізувати переваги масштабованості і передати ці переваги розробникам.

Amazon SimpleDB – інший веб сервіс, розроблений для того, щоб виконувати запити на структуровані дані Amazon Simple Storage Service (Amazon S3) в режимі реального часу. Цей сервіс працює в з'єднанні з Amazon EC2, щоб надати користувачам можливість зберігання, обробки і запитів наборів даних у межах навколишнього середовища хмари. Ці сервіси розроблені, щоб зробити веб масштабовані обчислення легше і більш прибутковими для розробників. Традиційно даний тип функціональності був забезпечений використанням кластерної реляційної бази даних, яка вимагає значних інвестицій. Впровадження даних технологій викликало більше складності і часто вимагало послуг адміністрування та підтримки бази даних.

У порівнянні з традиційним підходом, Amazon SimpleDB легка у використанні і забезпечує основну функціональність баз даних (наприклад, пошук в реальному часі та запити структурованих даних), що не наслідуючи складності експлуатації, що виникають при традиційному виконання. Amazon SimpleDB не вимагає схеми, дані індексуються автоматично, забезпечує простий інтерфейс API для зберігання і доступу до даних. Це позбавляє клієнтів від необхідності виконати завдання, такі як моделювання даних, обслуговування індексів і налаштування продуктивності.

Amazon Simple Queue Service (Amazon SQS) – сервіс приймає черги повідомлень для зберігання. При використанні Amazon SQS, розробники можуть просто перемістити дані, розподілені між компонентами своїх додатків, які виконують різні завдання, не втрачаючи при цьому повідомлення. При цьому досягається висока масштабованість і надійність. Amazon SQS працює як демонстрація масштабованої передачі повідомлень Amazon, інфраструктури як сервісу. Будь-який комп'ютер, пов'язаний з Інтернетом, може додати або прочитати повідомлення без необхідності в установці якого програмного забезпечення або спеціальний конфігурації брандмауера. Компоненти додатків, що використовують Amazon SQS, можуть запускатися незалежно і не повинні обов'язково розміщуватися в тій же самій мережі, що використовує ті ж самі технології або працюють в той же самий час.

Amazon CloudFront – веб-сервіс для доставки контенту (змісту). Amazon CloudFront інтегрується з іншими Amazon Web Services. Мета сервісу – дати розробникам і підприємствам простий спосіб поширювати контент для кінцевих користувачів з низькою затримкою, високою швидкістю передачі даних, при цьому не вимагаючи жодних зобов'язань. Запити об'єктів автоматично маршрутізіруются на найближча граничний сервер. Таким чином, зміст доставлено з кращою можливою продуктивністю. Граничний сервер отримує запит від комп'ютера користувача і з'єднується з іншим комп'ютером, викликаючи оригінальний сервер, де розташоване додаток. Коли оригінальний сервер виконує запит, він відправляє дані програми тому на граничний сервер, який передає дані назад комп'ютера клієнта, який виконував запит. Сервіс не є вільним для користування.

9.2. Платформа як Сервіс (PaaS)

Розвиток «хмарних» обчислень призвів до появи платформ, які дозволяють створювати і запускати веб-додатки. Платформа як сервіс (Platform as a Service, PaaS) – це надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб-додатків як послуги, організованої на основі концепції хмарних обчислень.

Модель PaaS створює всі умови необхідні для підтримки повного життєвого циклу створення і доставки веб-додатків і послуг доступних з мережі Інтернет, що не вимагають завантаження або установки програмного забезпечення для розробників, IT менеджерів або кінцевих користувачів. На відміну від моделі IaaS, де розробники можуть створювати певні примірники операційних систем з доморощеними додатками, розробники PaaS зацікавлені тільки веб розробкою і не піклуються про те, яка операційна система використовується. PaaS сервіси дозволяють користувачам зосереджуватися на інноваціях, а не на складній інфраструктурі. Організації можуть направити істотну частину їх бюджету на створення додатків, які забезпечують реальну цінність, замість витрат на підтримку інфраструктури. Модель PaaS таким чином відкриває нову еру масових інновацій. Тепер розробники в усьому світі можуть отримати доступ до необмеженої обчислювальної потужності. Будь-яка людина, що має доступ до Інтернету, може створювати додатки і легко розгортати.

Традиційний підхід створення і запуску локальних (On – Premises) додатків завжди був складний, дорогий і ризикований. Будівництво вашого власного рішення ніколи не надавало гарантії успіху. Кожен додаток було розроблено, щоб задовольнити певним діловим вимогам. Кожне рішення потребувало певної конфігурації апаратних засобів, операційної системи, бази даних, електронну пошту, веб-сервери, і т.д. Коли було створене середовище апаратного та програмного забезпечення, команда розробників повинна була вибрати комплекс платформ для розробки, щоб створювати додатки. Неминуче бізнес вимагає від розробників робити зміни в додатку. Змінений додаток вимагає нових циклів випробувальних робіт, перш ніж бути поширеним. Великі

компанії часто потребує спеціалізовані засоби, щоб розмістити їх в центрах обробки даних. Величезна кількість електрики необхідно для роботи серверів і підтримки системи кондиціонування. Нарешті, все це вимагає використання стійких до відмов майданчиків для центрів обробки даних так, щоб інформація могла копіюватися у разі збою.

PaaS пропонує більш швидку, більш економічно вигідну модель для розробки та доставки додатків. PaaS забезпечує всю інфраструктуру для запуску додатків через Інтернет. Аналогічні сервіси надають велику кількість компаній, таких як Microsoft, Amazon.com, Google. PaaS заснована на моделі обліку ліцензій чи моделлю підписки, таким чином, користувачі платять тільки за те, що вони використовують. Пропозиції PaaS включають робочі процеси для створення додатків, розробки додатків, тестування, розгортання і розміщення. Також сервіси додатків, віртуальні офіси, командне співробітництво, інтеграцію баз даних, безпека, масштабованість, зберігання, працездатність, управління станом, інструментарій приладових панелей і багато іншого.

Головні особливості PaaS включають сервіси для розробки, тестування, розгортання, розміщення та управління додатками для підтримки життєвого циклу розробки додатків. Веб інтерфейси інструментів створення, як правило, забезпечують деякий рівень підтримки щоб спростити створення користувацьких інтерфейсів, заснованих на таких технологіях як HTML, JavaScript та інших технологіях. Підтримка багатокористувацької архітектури допомагає уникнути проблем при розробці щодо використання додатків багатьма користувачами одночасно. Провайдери PaaS часто включають послуги для управління паралельною обробкою, масштабованість, відмовостійкість і безпекою. Інша особливість – це інтеграція з веб – службами і базами даних. Підтримка протоколу обміну структурованими повідомленнями в розподіленої обчислювальної середовищі (Simple Object Access Protocol, SOAP) та інших інтерфейсів дозволяють додаткам PaaS створювати комбінації веб – сервісів (які називають mashup) так само легко, як наявність доступу до баз даних і повторного використання послуг всередині приватних мереж. Здатність формувати та розповсюджувати код між спеціалізованими, зумовленими або розподіленими командами дуже збільшує продуктивність пропозицій вендорів PaaS. Інтегровані пропозиції PaaS забезпечують можливість для розробників, щоб найбільш добре розуміти внутрішню роботу їх додатків і поведінку користувачів при використанні інструментів, подібних приладової панелі, щоб розглянути внутрішні параметри, засновані на вимірах кількості паралельних з'єднань і т.д. Деякі пропозиції PaaS розширюють цей інструментарій, що дозволяє складати рахунки оплати за використання.

9.3. Microsoft Azure

Платформа корпорації Майкрософт Windows Azure (спочатку відома під назвою Azure Services Platform) – це група «хмарних» технологій, кожна з яких надає певний набір служб для розробників додатків. На рисунку 9.2 показано, що платформа Windows Azure може бути використана як додатками, що

виконуються в «хмарі», так додатками, що працюють на локальних комп'ютерах.

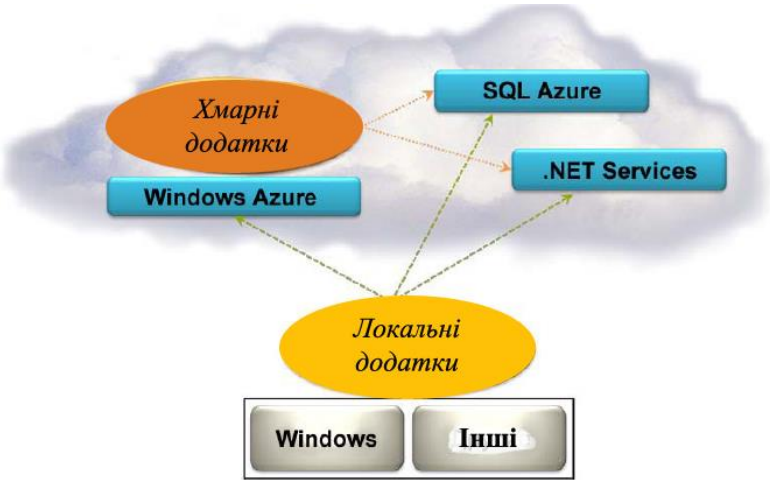


Рисунок 9.2. – Платформа Windows Azure підтримує програми, дані та інфраструктуру, що знаходяться в «хмарі»

Платформа Windows Azure складається з наступних компонентів:

– Windows Azure. Забезпечує середовище на базі Windows для виконання додатків і зберігання даних на серверах в центрах обробки даних корпорації Майкрософт.

– SQL Azure. Забезпечує служби даних в «хмарі»на базі SQL Server.

– NET Services. Забезпечують розподілену інфраструктуру для «хмарних»додатків і локальних додатків.

Кожен компонент платформи Windows Azure відіграє власну роль.

На високому рівні зрозуміти Windows Azure дуже легко. Це платформа для виконання додатків Windows і зберігання їх даних в Інтернеті («хмарі»). На рисунку 9.3 показані її основні компоненти.

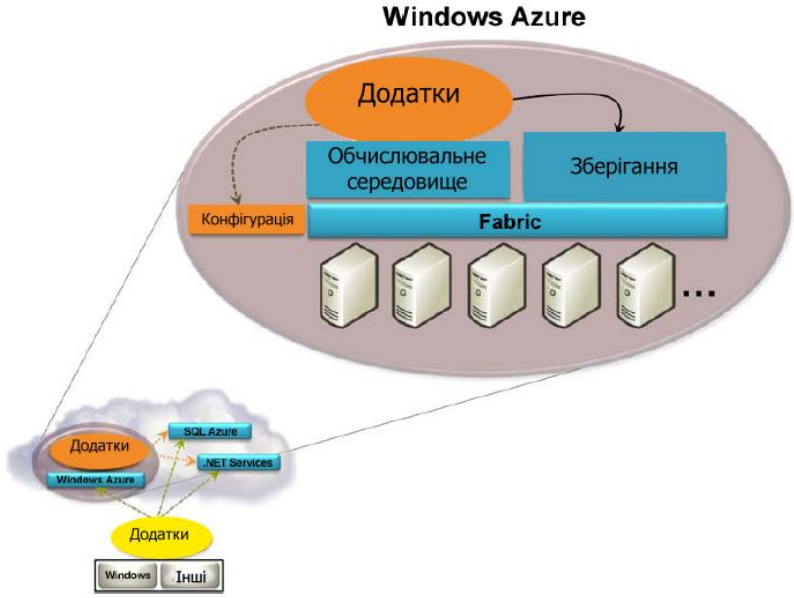


Рисунок 9.3. – Windows Azure надає «хмарним» програмам служби для обчислення і зберігання на базі Windows

Як показано на рисунку, Windows Azure виконується на великій кількості комп'ютерів, розташованих у центрах обробки даних корпорації Майкрософт, і доступний через Інтернет. Загальна структура підключення Fabric Windows Azure з'єднує безліч обчислювальних потужностей в єдине ціле. Служби Windows Azure з обчислення та зберігання побудовані на основі цієї структури.

Обчислювальна служба Windows Azur, працює на базі Windows. Для забезпечення первісної доступності цієї служби восени 2008 р. була відкрита для широкої публіки CTP-версія. Корпорація Майкрософт дозволила виконувати на Windows Azure тільки додатки, розроблені на платформі .NET Framework. Сьогодні, однак, Windows Azure також підтримує некерований код, дозволяючи розробникам виконувати програми, які розроблені не на базі .NET Framework. У будь-якому випадку такі додатки написані на звичайних мовах Windows – C#, Visual Basic, C++ та інших – за допомогою Visual Studio 2008 або інших засобів розробки. Розробники можуть створювати веб-додатки за допомогою таких технологій, як ASP.NET і Windows Communication Foundation (WCF), додатки, які виконуються як незалежні фонові процеси, або програми, що поєднують і те й інше.

Як додатки Windows Azure, так і локальні програми можуть отримувати доступ до служби сховища Windows Azure, роблячи це одним і тим же способом: за допомогою підходу RESTful. Однак Microsoft SQL Server не є базовим сховищем даних. Фактично сховище Windows Azure не відноситься до реляційних систем, і мова його запити не SQL. Оскільки воно спочатку призначено для підтримки додатків на базі Windows Azure, то забезпечує більш прості і масштабовані способи зберігання. Отже, воно дозволяє зберігати великі двійкові об'єкти (binary large object – blob), забезпечує створення черг для взаємодії між компонентами додатків і навіть щось на зразок таблиць з простою мовою запитів. (Для тих додатків Windows Azure, яким потрібна звичайне реляційне сховище, платформа Windows Azure надає базу даних SQL Azure, описану далі.)

Виконання програм і зберігання їх даних в Інтернеті має очевидні переваги. Наприклад, замість того, щоб купувати, встановлювати і експлуатувати власні комп'ютери, організація може довірити все це постачальнику послуг інтернету. При цьому замовники платять тільки за обчислювальні потужності і сховище, яке вони використовують, і не пов'язані з обслуговуванням великої кількості серверів, призначених тільки для пікових навантажень. Якщо додатки правильно написані, їх можна легко масштабувати, скориставшись перевагами величезних центрів обробки даних, які можуть запропонувати постачальники.

І все ж для отримання цих переваг потрібно ефективне управління. У Windows Azure кожен додаток має файл конфігурації, як показано на рис. 2. Змінюючи інформацію в цьому файлі вручну або за допомогою програми, власник додатку може контролювати різні аспекти його поведінки, такі як налаштування кількості примірників, які повинні виконуватися на платформі

Windows Azure. Структура Fabric платформи Windows Azure спостерігає за тим, щоб додаток підтримувалося в необхідному стані.

Щоб дозволити своїм замовникам створювати, налаштовувати програми та спостерігати за ними, Windows Azure надає портал, доступний за допомогою браузера. Замовник надає Windows Live ID, а потім вирішує, створювати йому обліковий запис розміщення для виконання додатків, обліковий запис зберігання для зберігання даних або і ту і іншу. Оплата використання програми замовниками може здійснюватися будь-яким зручним способом: за допомогою підписки, почасово або як-небудь інакше.

Windows Azure – це спільна платформа, яку можна використовувати в різних сценаріях. Наведемо кілька прикладів, всі вони описуються з урахуванням можливостей CTP-версії.

При створенні нового веб-сайту, скажімо, такого як Facebook, можна розробляти програми на платформі Windows Azure. Завдяки тому, що дана платформа підтримує як веб-служби, так і фонові процеси, додаток може надавати інтерактивний інтерфейс користувача і асинхронно виконувати роботу для користувачів. Замість того, щоб витрачати час і гроші, думаючи про інфраструктуру, пускова група може повністю зосередити свою увагу на розробці коду, який буде приносити користь користувачам і інвесторам. Компанія може також запустити невеликий веб – сайт, що вимагає незначних витрат, якщо у її додатку дуже мало користувачів. Якщо додаток набуває популярності і кількість користувачів зростає, Windows Azure при необхідності дозволяє масштабувати його.

Незалежні постачальники програмного забезпечення, що створюють версію програми як служби наявного локального додатку Windows, можуть розробити його на базі Windows Azure. Завдяки тому, що Windows Azure головним чином забезпечує стандартну середу Windows, переміщення бізнес-логіки програми на цю «хмарну» платформу не повинно створювати особливих проблем. Ще раз підкреслимо: розробка на базі наявної платформи дозволяє незалежним постачальникам ПЗ зосередити увагу на бізнес-логіці, тобто на тому, що дозволяє їм робити гроші, а не витрачати час на інфраструктуру.

Компанія, що створює додаток для своїх замовників, може вибрати для його розробки платформу Windows Azure. У силу того що Windows Azure підтримує .NET, не представляє труднощів знайти розробників з відповідними навичками до того ж за розумну плату. Виконання програми в центрах обробки даних Microsoft звільняє підприємства від відповідальності і витрат на підтримку власних серверів, перетворюючи капітальні витрати в експлуатаційні витрати. Особливо якщо у додатку є періоди пікового навантаження (якщо це, наприклад, мережевий магазин квітів, які необхідно вручити під час загального ажіотажу 8 березня), надання корпорації Microsoft функції підтримки великий серверної бази, необхідної для цього, може виявитися економічно вигідним.

Виконання додатків в «хмарі» – один з найважливіших аспектів «хмарних» обчислень. За допомогою Windows Azure корпорація Microsoft забезпечує як платформу для виконання додатків, так і спосіб зберігання даних.

У міру того, як зростає інтерес до «хмарних» обчислень, очікується створення ще більшої кількості програм Windows для цієї нової області.

Один з найбільш привабливих способів використання серверів, доступних через Інтернет, – це обробка даних. Мета SQL Azure – вирішити цю проблему, пропонуючи набір веб – служб для зберігання самої різної інформації і роботи з нею. У той час, як представники Microsoft заявляють, що поступово SQL Azure буде містити цілий ряд можливостей, орієнтованих на дані, включаючи створення звітів, аналіз даних та багато іншого, першими компонентами SQL Azure стануть база даних SQL Azure Database і засіб синхронізації даних Huron. Це наочно продемонстровано на рисунку 9.4.

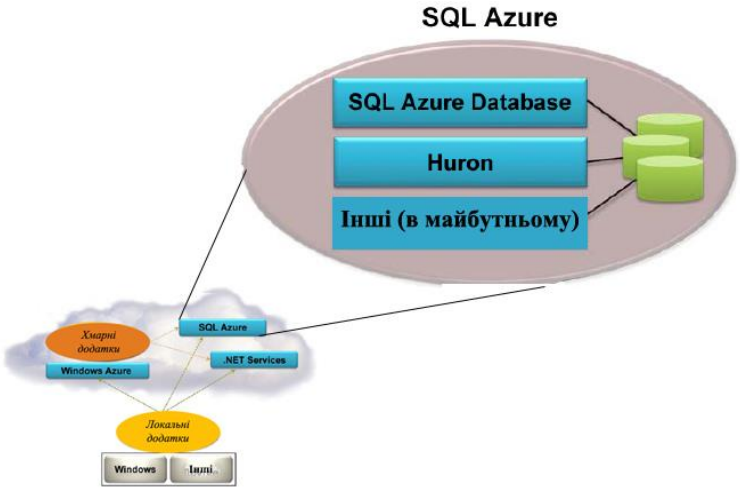


Рисунок 9.4. – SQL Azure обслуговує засоби в Інтернеті, орієнтовані на роботу з даними

База даних SQL Azure Database (раніше відома під назвою SQL Data Service) забезпечує систему управління базами даних (СКБД) в Інтернеті. Ця технологія дозволяє локальним і веб – додаткам зберігати реляційні та інші типи даних на серверах Microsoft в центрах обробки даних Microsoft. Так само як при роботі з іншими веб-технологіями, компанія платить тільки за те, що використовує, збільшуючи і зменшуючи обсяг використання (і витрати) у міру виникнення необхідності в змінах. Використання бази даних в «хмарі» також змінює характер капітальних витрат: на місце інвестицій у жорсткі диски і ПЗ для СУБД приходять експлуатаційні витрати.

На відміну від служби сховища Windows Azure база даних SQL Azure розроблена на основі Microsoft SQL Server. Тим неменш в первісній CTP – версії 2008 року база даних SQL Azure Database не надавала традиційний реляційний підхід до даних. Враховуючи відгуки замовників, корпорація Microsoft вирішила внести відповідні зміни. Надалі база даних SQL Azure Database буде підтримувати реляційні дані, забезпечуючи середу SQL Server в «хмарі» з індексами, уявленнями, збереженими процедурами, тригерами і багатьом іншим. Доступ до цих даних можна отримати за допомогою ADO.NET і інших інтерфейсів доступу до даних Windows. Фактично додатки, які сьогодні отримують доступ до SQL Server локально, працюватимуть майже точно так само з даними в SQL Azure Database. Для роботи з цією інформацією в «хмарі»

замовники можуть також використовувати локальне ПЗ, таке як служби звітів SQL Server.

У той час, як додатки можуть використовувати базу даних SQL Azure Database в значній мірі також, як локальну СУБД, вимоги до управління істотно скорочені. Замість того, щоб турбуватися про техніку, наприклад, забезпечувати моніторинг використання диска і обслуговування файлів журналу, замовник SQL Azure Database може зосередити увагу на тому, що дійсно важливо, на даних. Корпорація Microsoft буде відповідати за питання експлуатації. Крім того, так само як у випадку з іншими компонентами платформи Windows Azure, використання SQL Azure Database не складає труднощів. Потрібно просто зайти на веб – портал і надати необхідну інформацію.

Другий компонент SQL Azure був заявлений під назвою Huron Data Sync. Ця технологія, розроблена на основі Microsoft Sync Framework і SQL Azure Database, дозволяє синхронізувати реляційні дані в різних локальних СУБД. Власники даних можуть визначати, що саме має синхронізуватися, як повинні вирішуватися конфлікти і багато іншого.

Додатки можуть використовувати SQL Azure самими різними способами. Наведемо кілька прикладів:

– Додаток Windows Azure може зберігати дані в SQL Azure Database. Незважаючи на те, що Windows Azure забезпечує власне сховище, реляційні таблиці не входять до число пропонованих варіантів. Враховуючи те, що багато наявних програм використовують реляційне сховище, а багато розробників знають, як з ним працювати, значну кількість додатків Windows Azure, швидше за все, буде працювати з даними звичним способом, тобто з SQL Azure Database. Для підвищення продуктивності замовники можуть вказати, що певний додаток Windows Azure повинно виконуватися в тому ж центрі обробки даних, де SQL Azure Database зберігає інформацію цього додатка.

– У невеликій компанії або підрозділі великої організації додаток може використовувати SQL Azure Database. Замість того, щоб зберігати дані в базі даних SQL Server або Access, що працює на комп'ютері під чиїмось столом, додаток може використовувати переваги надійного і стійкого до відмов «хмарного» сховища.

– Припустимо, виробник хоче зробити інформацію про продукт доступною для своєї дилерської мережі і безпосередньо для замовників. Розміщення даних в SQL Azure Database дозволить зробити їх доступними для додатків, що виконуються на стороні дилерів, і для орієнтованих на замовників веб-додатків, які виконуються на стороні виробника.

– Компанія з клієнтською базою даних, реплицироваться в географічно віддалених місцях, повинна використовувати компонент Huron для синхронізації цих реплік. Можливо, в кожній з географічних точок потрібно власна копія даних для підвищення продуктивності, забезпечення доступності або з якихось інших причин. Автоматична синхронізація може зробити таке обов'язковий розподіл менш проблематичним.

Чи йде мова про програму Windows Azure, забезпеченні більшої доступності даних, синхронізації цих даних або про щось ще, служби даних в Інтернеті можуть виявитися дуже корисними. У міру появи нових технологій в рамках SQL Azure організації отримуватимуть можливість використання Інтернету для виконання все більшої кількості завдань, орієнтованих на роботу з даними.

Виконання програм і зберігання даних в Інтернеті відносяться до важливих аспектів обчислювальногомережевого середовища. Однак вони далеко не вичерпують його можливості. Інша можливість полягає в забезпеченні інфраструктури служб на базі «хмари», які можуть використовуватися локальними додатками або веб-додатками. Заповнити цю прогалину і покликані служби .NET Services.

Спочатку відомі як BizTalk Services, служби.NET Services пропонують функції для вирішення спільних проблем інфраструктури при створенні розподілених додатків. На рисунку 9.5 показані їх основні компоненти.

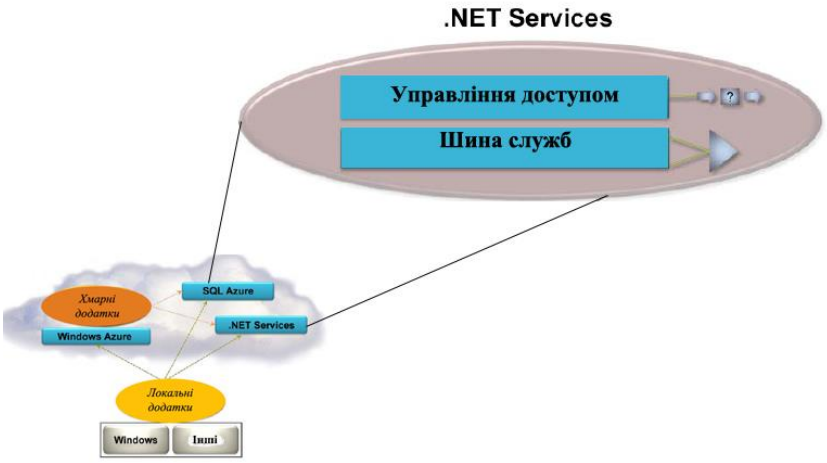


Рисунок 9.5. – Служби.NET Services забезпечують інфраструктуру в «хмарі», яка може бути використана для веб-додатків і локальних додатків

Служб NET Services складаються з наступних компонентів:

– Управління доступом. Все більше поширюється підхід до посвідчень, який полягає в тому, що кожен користувач повинен надати додатку маркер, що містить деякий набір тверджень. На підставі цих тверджень додаток вирішує, що раз – вирішено робити користувачеві. Ефективне здійснення цієї процедури в масштабах компанії вимагає федерації посвідчень, яка дозволяє приймати твердження, зроблені в одній області посвідчень, в іншій області. Може також знадобитися перетворення тверджень, що змінює їх при передачі з однієї області посвідчень в іншу. Служба управління доступом забезпечує реалізацію обох функцій на основі «хмари».

– Шина служб. Представлення служб додатків в Інтернеті набагато важче, ніж може здатися. Завдання шини служб – спростити цю процедуру, дозволяючи додаткам надавати кінцеві точки веб – служб, доступ до яких може бути отриманий іншими додатками – локальними або працюють в «хмарі». Кожній наданій кінцевій точці присвоюється URI, який клієнти можуть

використовувати для пошуку служби та отримання доступу до неї. Шина служб також вирішує проблему перетворення мережевих адрес і проходження через міжмережеві екрани без відкриття нових портів для наданих додатків.

Наведемо кілька прикладів використання служби .NET Services.

Незалежний постачальник ПЗ, який поставляє додаток, необхідний замовникам з різних організацій, може використовувати службу управління доступом для спрощення розробки та експлуатації програми. Наприклад, цей компонент .NET Services може перетворювати різні твердження, що застосовуються в різних організаціях з різними технологіями ідентифікації в узгоджений набір, відповідний для програми незалежного постачальника ПЗ. Таке перетворення дозволяє також розвантажити механізм федерації посвідчень за рахунок служби управління доступом, звільняючи незалежних постачальників ПЗ від необхідності виконання власної локальної програми федерації.

Припустимо, підприємство хоче відкрити доступ до одного з своїх додатків торговим партнерам. Воно може розподілити функції програми за допомогою веб-служб SOAP або RESTful і зареєструвати їх кінцеві точки за допомогою шини служб. Потім торгові партнери можуть використовувати шину для пошуку кінцевих точок і доступу до служб. Це дозволяє знизити ризики, пов'язані з наданням додатка, оскільки не вимагає відкриття нових портів в міжмережевим екрані компанії. Організація може також використовувати службу управління доступом, призначену для роботи з шиною служб, для раціоналізації відомостей про перевірку справжності, відправленої додатком її партнерами.

Так само як у випадку Windows Azure, надається портал, доступний за допомогою браузера, щоб дати замовникам можливість використовувати служби .NET Services за допомогою Windows Live ID. Мета корпорації Microsoft, що досягається за допомогою .NET Services, абсолютно очевидна: забезпечити корисну «хмарну» інфраструктуру для розподілених додатків.

9.4. Програмне забезпечення як Сервіс (SaaS)

Програмне забезпечення як сервіс (Software as a service, SaaS) або програмне забезпечення на вимогу (Software on Demand, SoD) – бізнес-модель продажу програмного забезпечення, при якій постачальник розробляє веб-додаток і самостійно управляє ним, надаючи замовникам доступ до програмного забезпечення через Інтернет. Основна перевага моделі SaaS для споживача полягає у відсутності витрат, пов'язаних з установкою, оновленням і підтримкою працездатності обладнання працюючого на ньому програмного забезпечення. Програмне забезпечення як сервіс є моделлю розповсюдження програмного забезпечення, в якій додатки розміщені у вендора SaaS або постачальника послуг і доступні для клієнтів по мережі, як правило, Інтернет. Модель SaaS доставки додатків стає все більш і більш поширеною технологією, яка підтримує веб-служби і сервіс-орієнтовану архітектуру (SOA). SaaS також часто асоційована з моделлю ліцензування, коли оплата відбувається в міру

отримання послуг. Тим часом, послуги широкосмугових мереж стали все більш і більш доступними, для підтримки доступу користувачів з більшої кількості місць по всьому світу.

Величезні успіхи, досягнуті постачальниками послуг Інтернету (ISP), щоб збільшити смугу пропускання і зберегти можливість використання більш потужних мікропроцесорів разом з недорогими пристроями зберігання даних. Це забезпечує величезну платформу для того, щоб проектувати, розгортати і використовувати програмне забезпечення через всі області бізнес і приватних обчислень. Додатки SaaS також повинні бути в змозі взаємодіяти з іншими даними й іншими додатками серед великого різноманіття навколишніх середовищ і платформ. Компанія IDC описує дві моделі поставки SaaS.

SaaS частіше за все призначений для забезпечення бізнес функціональності програмного забезпечення для корпоративних клієнтів за низькою ціною, що дозволяє позбутися від установки, управління, підтримки, ліцензування та високих витрат в компанії. Більшості клієнтам нецікаво знати, як або чому програмне забезпечення реалізовано, розгорнуто і т.д., але всі вони, в теж час, мають потребу у використанні програмного забезпечення в їх роботі. Багато типів програмного забезпечення добре задовольняють моделі SaaS (наприклад, бухгалтерський облік, робота з клієнтами, електронна пошта, облік трудових ресурсів, IT безпека, управління IT, відеоконференцзв'язок, веб-аналітика, управління веб – контентом). Різниця між SaaS і більш ранніми способами доставки додатків через Інтернет в тому, що рішення SaaS були розроблені спеціально, щоб працювати з веб браузерами. Архітектура додатків на основі SaaS спеціально призначена для підтримки обробки запитів від великої кількості користувачів. У цьому й полягає велика різниця між традиційним клієнт-серверним додатком рішенням, розташованим у постачальників послуг. З іншого боку, постачальники послуг SaaS збільшують економію масштабування при розгортанні, керуванні, підтримці та обслуговуванні їх пропозицій.

Багато типів компонентів програмного забезпечення і фреймворків можуть бути використані при розробці додатків SaaS. Використовуючи нові технології в цих сучасних компонентах і середовищах розробки додатків, можна значно зменшити час розробки та вартості перетворення традиційного продукту в рішення SaaS. Згідно Microsoft, SaaS архітектура може бути класифікована в один з чотирьох рівнів, з ключовими ознаками: простота конфігурації, ефективність при багатокористувацькому доступі і масштабованість. Кожен рівень відрізняється від попереднього додаванням однієї з цих ознак. Розглянемо рівні, описані Microsoft:

– Архітектурний Рівень 1 – Спеціальний/Настроюваний.

Перший рівень є фактично найнижчим. Кожен клієнт має унікальну, налаштовану версію розміщуваного додатку. Додаток запускає свої власні екземпляри на серверах. Міграція традиційних немережевих або клієнт – серверних додатків на цей рівень SaaS, як правило, вимагає незначних зусиль при розробці та зменшує експлуатаційні витрати, завдяки об'єднанню серверного апаратного забезпечення та адміністрування.

– Архітектурний Рівень 2 – Конфігурування.

Другий рівень SaaS забезпечує велику гнучкість програми завдяки метаданих конфігурації. На даному рівні клієнти можуть використовувати багато окремих екземплярів однієї програми. Це дозволяє вендорам задовольняти змінні потреб кожного клієнта при використанні деталізованої конфігурації. Також полегшується обслуговування, з'являється можливість оновити загальну кодову базу.

– Архітектурний Рівень 3 – Ефективність мульти орендатора.

Третій рівень відрізняється від другого наявністю підтримки багатокористувацького доступу. Єдиний екземпляр програми здатний обслужити всіх користувачів. Даний підхід дозволяє більш ефективно використовувати ресурси сервера непомітно для кінцевого користувача, але, в кінцевому рахунку, цей рівень не дозволяє виконувати масштабування системи.

– Архітектурний Рівень 4 – Масштабованість.

У четвертому рівні SaaS, масштабованість добавлена завдяки використанню багаторівневої архітектури. Ця архітектура здатна підтримувати розподіл навантаження ферми ідентичних екземплярів додатків, запущених на змінному кількості серверів, яке досягає сотень і навіть тисяч. Потужність системи може бути динамічно збільшена або зменшена відповідно до вимог. Це здійснюється шляхом додавання або видалення серверів без необхідності для подальшої зміни прикладної архітектури програмного забезпечення.

Розгортання додатків в сервіс-орієнтованій архітектурі є більш складною проблемою, ніж розгортання програмного забезпечення в традиційних моделях. В результаті вартість використання програми SaaS грунтується на числі користувачів, які здійснюють доступ до сервісу. Досить часто виникають додаткові витрати, пов'язані з використанням послуг сервісної служби, додаткової смуги пропускання, і додаткового дискового простору. Доходи постачальників послуг SaaS зазвичай спочатку нижче, ніж традиційні витрати за ліцензії на програмне забезпечення. Однак компроміс для більш низьких витрат ліцензії – щомісяця повертає дохід, який розглядається фінансовим директором компанії, як більш передбачуваний критерій існування бізнесу. До ключових особливостей програмного забезпечення SaaS відносяться:

– Управління по мережі і мережевий доступ до комерційного програмного забезпечення у централізованих центрах обробки даних, а не на сайтах клієнтів, надання можливості клієнтам отримати доступ до додатків віддалено через Інтернет.

– Доставка додатків по моделі «один до багатьох», на противагу традиційної моделіодин до одного».

– Централізована модернізація та оновлення, що дозволяє уникнути необхідності завантаженняі встановлення додатків користувачем. SaaS часто використовується у великих мережах комунікацій і програмного забезпечення для спільної роботи, іноді як програмне розширення до архітектури PaaS.

Цикли розробки програм в компаніях можуть займати досить довгий час, споживаючи великі ресурси і приводячи до незадовільних результатів.

Хоча рішення втратити контроль є важким, це може привести до поліпшення ефективності, зниження ризиків і скорочення витрат. Постійно збільшується число компаній, які хочуть використовувати модель SaaS для корпоративних додатків, таких як робота з клієнтами, фінансові витрати, управління персоналом. Модель SaaS гарантує підприємствам, що всі користувачі системи використовують правильну версію програми і тому формат зареєстрованих та переданих даних коректний, сумісний і точний. Покладаючи відповідальність за програми на постачальника SaaS, підприємства можуть зменшити витрати на адміністрування та управління, які необхідні для підтримки власного корпоративного програми. SaaS збільшує доступність додатків в мережі Інтернет. SaaS гарантує, що всі транзакції додатки зареєстровані. Переваги SaaS для клієнтів досить зрозумілі:

- Раціональне управління.
- Автоматизоване оновлення та виправлення.
- Цілісність даних в рамках підприємства.
- Спільна робота співробітників підприємства.
- Глобальна доступність.

Серверна віртуалізація може використовуватися в архітектурі SaaS замість або на додаток до підтримки багато режиму. Головна перевага платформи віртуалізації – збільшення продуктивності системи без необхідності в додатковому програмуванні. Ефект об'єднання спільного використання ресурсів і платформи віртуалізації в рішення SaaS забезпечує більшу гнучкість і продуктивність для кінцевого користувача.

9.5. Комунікація як Сервіс (CaaS)

Комунікація як Сервіс (CaaS) – побудоване в хмарі комунікаційне рішення для підприємства. Постачальники цього типу хмарного рішення відповідають за управління апаратним та програмним забезпеченням, необхідним для того, щоб надати:

- систему зв'язку, що забезпечує передачу мовного сигналу по мережі Інтернет або по будь-яким іншим IP-мережах (VoIP),
- обмін миттєвими повідомленнями (IM),
- відеоконференц-зв'язок.

Ця модель почала свій еволюційний процес в індустрії телекомунікацій, не сильно відрізняючись від моделі SaaS, стала результатом сектора служб доставки програмного забезпечення. Вендори CaaS відповідальні за управління апаратним та програмним забезпеченням їх користувачів. Вендори CaaS, як правило, надають гарантію за якість обслуговування (QoS) у відповідності з угодою сервісного обслуговування (SLA).

Модель CaaS дозволяє діловим клієнтам вибірково розгортати засоби комунікацій і послуг на підставі оплати послуг в строк для використовуваних сервісів. CaaS розроблений на ціновій політиці загального призначення, яка надає користувачам всебічний, гнучкий і легкий у розумінні сервісний план.

Згідно Gartner, ринок CaaS, як очікується, буде нараховувати $ 2,3 мільярда в 2011 році, з щорічним темпом зростання більше 105 %.

Сервісні пропозиції CaaS часто пов'язані і включають інтегрований доступ до традиційного голосу (або VoIP) і даних, додаткова функціональність об'єднаних комунікацій, такі як відео виклики, спільна робота, бесіди, присутність в реальному часі і передача повідомлень, телефонна мережа, місцева і розподілена голосові послуги, голосова пошта. CaaS рішення включає надлишкове перемикання, мережа, надмірність обладнання, WAN failover – що безумовно підходить до потреб клієнтів. Всі транспортні компоненти VoIP розташовані в географічно розподілених, безпечних інформаційних центрах для високої доступності і життєздатність. CaaS припускає гнучкість і масштабованість для дрібного і середнього бізнесу, чого найчастіше самі компанії не можуть забезпечити. Постачальники послуг CaaS підготовлені до пікових навантажень, надають послуги з розширення ємності пристроїв, станів чи області покриття на вимогу замовника. Пропускна здатність мережі та набори засобів можуть бути змінені динамічно, таким чином, функціональність йде в ногу зі споживчим попитом і ресурси, що знаходяться у власності постачальника не використовуються даремно. На відміну від постачальника послуг, перспектива клієнта фактично не приводить до ризику обслуговування застарілого обладнання, так як обов’язок постачальника послуг CaaS полягає в тому, щоб періодично модернізувати або замінювати апаратне і програмне забезпечення, щоб підтримувати платформу в технологічно актуальному стані.

CaaS не вимагає контролю від клієнтів. Це позбавляє від необхідності клієнтів вчиняти будь капіталовкладення в інфраструктуру, і це усуває накладні витрати для інфраструктури. З рішенням CaaS клієнти в стані посилювати комунікаційні послуги класу підприємства, не маючи необхідності до побудови власне рішення всередині своєї організації. Це дозволяє клієнтам перерозподіляти бюджет і трудовитрати персоналу, використовувати їх у тих місцях, де це найбільш необхідно.

Від телефонної трубки, яку можна знайти на столі кожного співробітника до клієнтського програмного забезпечення на ноутбуці співробітника, VoIP приватна основа, і всі необхідні дії між кожним з компонентів у вирішенні CaaS підтримуються в режимі 27/7 постачальником послуг CaaS.

– Рішення розміщення та управління.

Віддалене управління послугами інфраструктури забезпечується третіми особами, здавалося неприпустимою ситуацією для більшості компаній. Однак за минуле десятиліття з розвитком технологій, організацією мережі та програмним забезпеченням ставлення змінилося. Це частково пов'язано зі зниженням витрат при використанні обраних послуг. Однак на відміну від одиничних послуг пропозицію постачальників послуг CaaS надає повне комунікаційне рішення, яке є повністю кероване одним вендором. Поряд з особливостями, такими як VoIP і об'єднані комунікації, інтеграція офісної автоматичної телефонної станції з додатковою функціональністю управляється

одним вендором, який відповідальний за всю інтеграцію і доставку послуг користувачам.

– Зручність керування та функціональність.

Коли клієнти користуються послугами зв'язку на стороні постачальника послуг CaaS, вони платять тільки за необхідну функціональність. Постачальник послуг може розподіляти вартість послуг. Як зазначалося раніше, це сприяє більш економічному впровадження та використання загальної необхідної функціональності для клієнтів. Економія за рахунок зростання виробництва дозволяє постачальникам послуг здійснювати обслуговування досить гнучко, вони не прив'язані до єдиному постачальнику інвестицій. Постачальники послуг в змозі посилити рішення найкращих серед аналогічних постачальників, таких як Microsoft, Google, Amazon, Cisco, Nortel більш економічно.

– Немає витрат коштів на обладнання.

Все обладнання розташоване у постачальників послуг CaaS, це фактично позбавляє від необхідності клієнтів підтримувати власні інформаційні центри і обладнання. Відсутні витрати коштів на електроспоживання, охолодження, оренду приміщень. Клієнти отримують багаторазову вигоду, використовуючи центри обробки даних масштабу великих авіакомпаній з повним резервуванням – і це все включено в щомісячну оплату.

– Гарантована безперервність бізнесу.

Чи дозволяє Ваш план аварійного відновлення після катастрофічних подій в центрі обробки даних продовжувати безперервно працювати Вашому бізнесу? Як довго Ваша компанія може працювати при відключенні електроенергії? Для більшості компаній ці події неминуче означають відчутні фінансові втрати, пов'язані з простоєм бізнесу. Розподіл інформаційної системи компанії між географічно розподіленими центрами обробки даних стають нормою для все більшого числа компаній. Це пом'якшує ризик фінансових втрат і дозволяє компаніям розташованим в місці, де сталися якісь катастрофічні події, відновлювати інфраструктуру так скоро, наскільки це можливо. Цей процес здійснений постачальниками послуг CaaS. Для великої кількості компаній, що працюють з голосовою передачею даних, перебої в роботі системи є катастрофічними. На відміну від цілісності даних, усунення єдиних точок відмови для голосового мережі є звичайно досить дорогими через великий масштаб і складність управління проектом. Розв'язки CaaS володіють багаторазовими рівнями надмірності системи, що виключає із системи єдині точки відмови.

9.6. Моніторинг як Сервіс (MaaS)

Моніторинг як Сервіс (Monitoring-as-a-Service, MaaS) забезпечує в хмарі безпеку, насамперед на бізнес платформах. За минуле десятиліття MaaS став все більш і більш популярним. З появою хмарних обчислень, популярність MaaS стала більше. Контроль безпеки зачіпає захист клієнтів – підприємств або уряду від кібер загроз. Служба безпеки відіграє важливу роль у забезпеченні та підтримці конфіденційнjсті, цілісністі, і доступності засобів IT. Однак час і

обмежені ресурси обмежують заходи безпеки та їх ефективність для більшість компаній. Це вимагає постійної пильності безпеки інфраструктури і критичних інформаційних засобів. Багато промислових правил вимагають, щоб організації контролювали своє середовище безпеки, журнали серверів, та інші інформаційні засоби, щоб гарантувати цілісність цих систем. Однак забезпечення ефективного контролю стану безпеки може бути складним завданням, тому що воно вимагає передових технологій, кваліфікованих експертів з безпеки, і масштабовані процес, жоден з яких не є дешевим. Сервіси контролю стану безпеки MaaS пропонує контроль в реальному часі, в режимі 24/7 і практично негайні реагування по інцидентах через інфраструктуру безпеки. Ці сервіси допомагають захистити критичні інформаційні активи клієнтів. До появи електронних систем забезпечення безпеки, контроль стану безпеки і реагування залежали у великій мірі від людських ресурсів і людських здібностей, які обмежували правильність і ефективність контролюючих зусиль. За минулі два десятиліття, були розроблені інформаційні технології в системах забезпечення безпеки, які здатні взаємодіяти з центрами операційної безпеки (SOC) через корпоративні мережі, що значно змінило картину. Дані засоби включають дві важливі речі:

– Загальна вартість володіння центром операційної безпеки набагато вища, ніж для сучасної технології SOC.

– Досягнення більш низьких операційних витрат безпеки і більш висока ефективність засобів безпеки.

SOC послуги контролю стану безпеки можуть поліпшити ефективність інфраструктури безпеки клієнта, активно аналізуючи журнали та оповіщення від пристроїв інфраструктури цілодобово і в режимі реального часу. Контроль команд співвідносить інформацію з різних пристроїв безпеки, щоб надати аналітикам з безпеки даних, необхідні їм для усунення помилкових загроз і для реагування на реальні загрози підприємства. Служба інформаційної безпеки може оцінити продуктивність системи на періодично повторюваній основі і забезпечити рекомендації для удосконалень якщо необхідно.

Сервіс раннього виявлення повідомляє про нові слабкі місця в безпеці незабаром після того, як вони з'являються. Взагалі, загрози взаємопов'язані з джерелами, що мають відношення до третьої сторони. Звіт звичайно посилається по електронній пошті відповідальній людині, призначеній компанією. Звіти про вразливість безпеки, окрім вмісту докладного опису уразливості, також включає інформацію про вплив даної уразливості на систему або програму. Найбільш часто звіт також вказує на певні дії, які потрібно виконати, щоб мінімізувати ефект вразливості.

Платформа, управління та моніторинг сервісу часто надаються як приладова панель, що дозволяє в будь-який час дізнатися робочий стан системи. Доступ можна отримати через веб-інтерфейси, що дозволяє працювати віддалено. Кожен робочий елемент, який перевіряється зазвичай містить робочий індикатор статусу, завжди беручи до уваги критичний вплив кожного елемента. Дані сервісу дозволяють визначити, які елементи знаходяться в робочому стані, яким не вистачає потужності, а які знаходяться за межами

встановлених параметрів. Виявляючи і ідентифікуючи такі проблеми, можна вживати профілактичні заходи, для запобігання втрати працездатності сервісу.

Ключові терміни:

Інфраструктура як Сервіс, IaaS – надання комп'ютерної інфраструктури (як правило, це платформи віртуалізації) як сервісу.

Платформа як сервіс, PaaS – надання інтегрованої платформи для розробки, тестування, розгортання і підтримки веб-додатків як послуги, організованої на основі концепції хмарних обчислень

Програмне забезпечення як сервіс, SaaS – бізнес-модель продажу програмного забезпечення, при якій постачальник розробляє веб-додаток і самостійно управляє ним, надаючи замовникам доступ до програмного забезпечення через Інтернет.

Комунікація як Сервіс, CaaS – побудоване в хмарі комунікаційне рішення для підприємства MaaS.

Література:

1. John W. Rittinghouse, James F. Ransome – «Cloud Computing: Implementation, Management, and Security»

ЛЕКЦІЯ 10. ХМАРНІ СЕРВІСИ VMWARE ТА GOOGLE

Google Apps – це середовище, яке надає такі засоби спільної роботи: вже став популярним поштовий сервіс GMail, клієнт обміну миттєвими повідомленнями Google Talk (фактично сервіс повністю придатний для спілкування з будь-яким jabber-користувачем), календар GoogleCalendar, засоби для роботи з документами і електронними таблицями Google Docs & Spreadsheets, «центральну сторінку» - місце для зручного розміщення тієї інформації, яка буде спільною для всіх користувачів, редактор сторінок від Google, який дозволяє швидко створити і опублікувати потрібну інформацію.

10.1. Функції, доступні користувачеві

Описувати всі функції, які доступні кожному користувачеві системи Google Apps, – досить тривале заняття, так як система поєднує в собі кілька досить гнучких продуктів від Google, кожен з яких вже успішно зарекомендував себе. Крім того, більшість з цих сервісів досить популярні окремо.

Для початку відзначимо, що *Google Apps* – це серйозний сервіс, який має декілька пакетів, кожен з яких відрізняється різною кількістю послуг, що надаються Google. Всі пакети надають: повний набір сервісів (пошта, календар, робота з документами, створення сторінок, клієнт обміну швидкими повідомленнями і так далі), відсутність обмежень на кількість користувацьких облікових записів, доступ з мобільних пристроїв, панель керування адміністратора. Таким чином, базовий набір, який надається всім користувачам, приблизно однаковий, крім того, існує система міграції з одного пакету на інший, яка дозволяє почати з простішого і безкоштовного пакету, а пізніше по необхідності мігрувати на відповідний пакет з потрібними доповненнями. Основні відмінності в наступному.

Починати краще зі стандартного пакету Google Apps. Він безкоштовний.

– Стандартний пакет*(StandardEdition)* – 2 гігабайти вільного місця під пошту, допомога через онлайн-ресурси (але не онлайнова телефонна допомога), присутність контекстної реклами на сторінках сервісів.

– Прем'єр-пакет*(PremierEdition)* – 10 гігабайт під пошту, 99,9% гарантований uptime поштової служби, можливість управління ресурсами, онлайн-допомога в режимі 24/7, яка включає телефонні консультації, API для того, щоб найкращим чином інтегрувати Google Apps у вже існуючу інфраструктуру. Єдиний з усіх пакетів, який не є безкоштовним. Вартість даного пакету залежить від кількості призначених для користувачів облікових записів. Користувачі пакета отримують доступ до всіх нових функцій та сервісів системи в міру їх виходу, наприклад, у найближчих планах сервіс міграції з інших поштових клієнтів, який дозволить виконати міграцію з найменшими зусиллями.

– Пакет для освітніх установ*(EducationEdition)* – все те ж саме , що і в попередньому випадку. З невеликою різницею: всього 2 гігабайти на один

обліковий запис, відсутність гарантії 99,9% часу безперебійної роботи. Пакет надається за окремою ліцензією для некомерційних освітніх організацій.

Після того як адміністратор, який працює з Google Apps, додав користувача в систему (домен Google Apps), користувач може працювати з усіма сервісами, які включені в систему. Однією з основних ідей Google Apps є глобальна інтеграція всіх сервісів і організація зручної роботи для людей, які об'єднані спільною системою Google Apps. Це дозволяє серйозно економити час на організації спільної роботи, оскільки середовище для роботи виявляється готовою до використання вже через лічені хвилини після повної активації.

10.2 Пошта та обмін повідомленнями

Після того як поштовий сервіс, що відноситься до домену, активізований, він буде доступний користувачам на спеціальній сторінці, ім'я якої визначається системним адміністратором. Якщо налаштування домену дозволяють це, то всі користувачі домену автоматично будуть додані в контакти кожного нового користувача системи.

Рисунок 10.1. – Список контактів.
Всі користувачі домену автоматично присутні в адресній книзі

Аналогічно і з клієнтом обміну швидкими повідомленнями: нові користувачі домену автоматично будуть додані в контакт-лист клієнта. Це забезпечує миттєвий старт роботи користувача після реєстрації його в системі. Вже немає необхідності додавати в контакт-лист кожного колегу окремо.

Очевидно, що користувачам дістаються і всі стандартні можливості поштового сервісу від Google: архівація пошти, фільтрація спаму, можливість пошуку по всім поштовим повідомленням, створення фільтрів, доступ через POP, пересилання пошти та багато інше. Те ж саме можна сказати і про Google Talk – всі функції доступні повністю, причому адміністратор може обмежити можливість користувачів додавати в контакт-лист користувачів з інших доменів або з інших jabber-серверів, що дозволяє обмежити коло спілкування тільки необхідними контактами.

10.3. Календар

Дуже зручний засіб планування особистого робочого часу, який в умовах глобальної інтеграції дозволяє планувати не тільки свій робочий час, а й враховувати робочий час і завдання колег. Основні можливості календаря в Google Apps: створення подій, для кожного з яких можна визначити назву події, час і тривалість, визначення складу учасників і перевірка їх зайнятості під час

події, установка нагадувань про події, перегляд чужих календарів, робота з календарем на мобільних пристроях, управління доступом до календарів і так далі.

Рисунок 10.2. – Перегляд подій в календарі іншого користувача

Користувач домену може додати календар конкретного співробітника, використовуючи спеціальну форму, яка дозволяє шукати календарі, використовуючи як ключові слова, так і адресу електронної пошти співробітника. Таким чином, можна завжди мати актуальну інформацію про завдання та роботі певного співробітника (очевидно, що тільки в тій мірі, в якій це дозволяє визначити його календар).

10.4. Робота з документами

Підтримуються всі популярні формати документів: Word, Excel, OpenOffice.

В даний час доступна можливість роботи з наступними типами файлів: документи Word і Excel, документи OpenOffice, RTF, HTML і текстові документи. Такий набір підтримуваних форматів забезпечує придатність сервісу для широкого кола користувачів. Результати роботи можуть бути збережені як на локальний комп'ютер, так і залишені на зберігання на сервері. Таким чином, для повноцінної роботи з документами досить просто мати доступ до сервісу Google Apps з будь-якої точки світу, з будь-якого комп'ютера.

Порівняння можливостей самих редакторів Google Docs & Spreadsheets з Word і Excel або OpenOffice є темою для окремої дискусії. Практика використання показує, що функцій достатньо для того, щоб підготувати нормальний документ, який містить зазвичай використовувані елементи оформлення: списки, форматування і різні стилі, таблиці, зображення та гіперпосилання і так далі.

Google Docs & Spreadsheets надає широкі можливості для спільної роботи над документами . Для того щоб зробити роботу з документами як можна більш зручною і продуктивною для групи співробітників, доступні наступні можливості:

– Управління версіями документа. Проміжні версії документа створюються системою автоматично досить часто і, крім того, кожного разу, коли користувач зберігає документ. Доступна функція порівняння двох обраних версій, що дозволяє легко відслідковувати зміни, які були внесені черговим редагуванням документа.

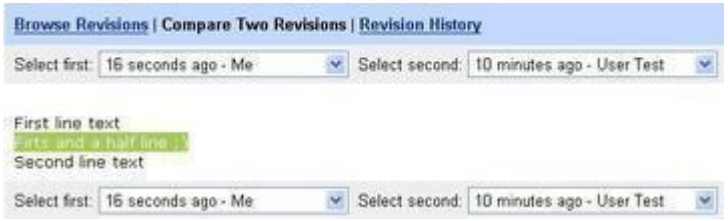


Рисунок 10.3. Порівняння версій Зеленим кольором виділено зміни між версіями документа

– Управління доступом до документа. Можна запрошувати користувачів системи до спільної роботи з документом, вказуючи, які права даються користувачеві: тільки перегляд документа або редагування. Додаткові можливості дозволяють користувачеві, запрошеному працювати з документом, у свою чергу запрошувати інших користувачів. Люди, які одночасно працюють з документом, можуть влаштовувати чат для обговорення змін в документі, які будуть видні всім учасникам обговорення. Доступна можливість публікації документа з постійною адресою, що дає можливість будь-якому співробітникові домену отримувати доступ до документа (наприклад, після того, як всі зміни зроблені і схвалені, документ публікується для загального ознайомлення). Крім того, існує можливість архівації документів, які вже приведені в їх фінальне стан, але ще можуть знадобитися.

Рисунок 10.4. – Запрошення користувачеві на прочитання документа

Спільна робота з документами в Google Apps організована на високому рівні, достатнім для того, щоб працювати з документами і швидко та ефективно отримувати доступ до загальної інформації. А для користувача функції редагування вмісту документів лише в окремих аспектах поступаються звичним Word, Excel і OpenOffice.

10.5. Стартова сторінка і редактор сторінок

Стартова сторінка – перше, що побачать користувачі після входу в Google Apps.

Стартова сторінка – це те місце, яке призначене для того, щоб бути першим, що побачать користувачі після входу в систему Google Apps. Ця сторінка схожа з персональною сторінкою Google*(www.igoogle.com),* так само вона призначена бути першою сторінкою, з якою користувач стикається, починаючи роботу. На ній можуть бути розташовані гаджети від Google і сторонніх розробників, а також інформація, необхідна для початку роботи співробітників. Далеко не повний список корисних речей, які можна розмістити на стартовій сторінці Google Apps: гаджети для попереднього перегляду особистих поштових скриньок і подій у календарі, пошук, а також можна використовувати і спеціальний Google Custom Search, який дозволяє шукати тільки те, що реально необхідно, перегляд RSS, закладки, Google Notebook і ще кількість корисних елементів.

Редактор веб-сторінок дозволяє легко і швидко створювати власні сторінки, які потім зручно публікувати за допомогою сервісу Google. Для створення можна використовувати велику кількість вже готових дизайнів (розташування елементів на сторінці, колірна гамма і так далі) і досить зручний редактор, який практично не вимагає від користувача знання HTML, CSS і інших мов. Публікація нової сторінки відбувається миттєво, що істотно скорочує час, який витрачається на те, щоб донести інформацію до решти користувачів або клієнтів.

Інтеграція в Google Apps досягла того, що документи, які прийшли вам поштою, вже відразу з поштової скриньки можна відкрити для роботи в Google Docs & Spreadsheets. Перемикання між різними сервісами здійснюється простим кліком мишки. Втім, немає проблем у тому, щоб тримати їх всі відкритими і готовими для роботи в різних вікнах браузера.

Таким чином, Google Apps створює середовище, яке зручне не тільки для спільної роботи кількох людей, які знаходяться в різних місцях, але і для організації спільної роботи великої кількості співробітників, які можуть бути розкидані по різних країнах. Враховуючи те, що компанія Google досить часто надає API до своїх продуктів, тим самим дозволяючи створювати власні додатки, які ще більш інтегрують додатки Google в конкретне середовище роботи, можна сказати, що Google Apps може стати дуже зручним середовищем для спільної роботи. Крім того, компанія Google постійно веде роботу над поліпшенням існуючих сервісів і додаванням нових. У найближчих планах додавання сервісу для міграції з різних поштових клієнтів на GMail і можливість створювати і редагувати презентації, тобто сервіс, аналогічний програмі PowerPoint.

10.6. App EngineApp Engine

Google дозволяє виконувати ваші веб-додатки в інфраструктурі Google. Додатки App Engine легко створювати, підтримувати та вдосконалювати в міру збільшення трафіку і сховища даних. При роботі з App Engine вам не знадобиться підтримувати сервер: просто завантажте свій додаток, і користувачі зможуть працювати з ним.

Додаток можна опублікувати у власному домені (наприклад, *http://www.example.com/)* за допомогою Служб Google. Або скористатися безкоштовним ім'ям в домені appspot.com. Додаток можна зробити доступним для всіх або надати доступ тільки учасникам вашого колективу.

Google App Engine підтримує програми, написані на декількох мовах програмування. Завдяки середовищу виконання Java App Engine можна створювати додатки за допомогою стандартних технологій Java, в тому числі JVM, сервлетів Java і мови програмування Java, або іншої мови, що використовує інтерпретатор або компілятор на JVM, наприклад JavaScript або Ruby. Крім того, App Engine надає спеціальне середовище виконання Python, яке включає швидкий інтерпретатор і стандартну бібліотеку Python. Середовища виконання Java і Python розроблені спеціально для того, щоб програми могли швидко і безпечно виконуватися без взаємодії з іншими додатками в системі.

З App Engine потрібно платити тільки за те, чим ви користуєтесь. Не потрібно платити за установку або вносити періодичні платежі. Оплата за використані додатком ресурси, такі як обсяг зберігання і трафік, вимірювані в гігабайтах, стягуються по обгрунтованих ставках. Можна керувати максимальною кількістю ресурсів, які додаток може використовувати, що дозволить завжди залишатися в межах бюджету.

Почати використовувати App Engine можна безкоштовно. Вам не доведеться платити за програми, що використовують менше 500 МБ сховища, а також ресурси ЦП і трафік, достатні для ефективного додатку, який обслуговує до п'яти мільйонів переглядів сторінок на місяць. Включивши оплату для програми, ці обмеження підвищуються, а оплата стягується тільки за ресурси, використані понад безкоштовних рівнів.

10.7. Середовище додатків

Google App Engine дозволяє легко створювати додатки, які надійно працюють навіть при великому навантаженні і з великими обсягами даних. App Engine включає наступні функції:

– динамічна робота в Інтернеті з повною підтримкою основних веб-технологій;

– постійне сховище із запитами, сортуванням і транзакціями;

– автоматичне масштабування і регулювання навантаження;

– API для аутентифікації користувачів і відправка електронної пошти за допомогою акаунтів Google;

– повнофункціональне локальне середовище розробки , що імітує Google App Engine на вашому комп'ютері;

– заплановані завдання для відстеження подій в певний час або через регулярні інтервали.

Додаток може виконуватися в одному з двох середовищ виконання: Java і Python. Кожне середовище надає стандартні протоколи і основні технології для розробки веб-додатків.

Програми працюють в безпечному середовищі, що забезпечує обмежений доступ до прикладної операційної системи. Обмеження дозволяють App Engine поширювати веб-запити для програми на кілька серверів, а також запускати і зупиняти сервери в залежності від трафіку. Тестове середовище ізолює ваш додаток у власному безпечному та надійному середовищі, незалежному від обладнання, операційної системи та фізичного розташування веб-сервера.

Нижче наведені приклади обмежень надійної тестового середовища.

Додаток може отримувати доступ до інших комп'ютерів в Інтернеті тільки через надані API, службу отримання даних по URL і службу електронної пошти. Інші комп'ютери можуть підключатися до додатка тільки шляхом HTTP-запитів (або HTTPS-запитів) на стандартних портах.

Додаток не може виконувати запис у файлову систему. Додаток може зчитувати файли, але тільки ті, які були завантажені разом з кодом програми. Додаток повинен використовувати сховище даних App Engine, кеш пам'яті і інші служби для всіх даних, що зберігаються, між запитами.

Код програми виконується тільки у відповідь на веб-запит або завдання Cron і в будь-якому випадку повинен повертати дані відповіді протягом 30 секунд. Оброблювач запитів не може створити підпроцесс або виконувати код після відправки відповіді.

Для середовища виконання Java можна розробити програму за допомогою стандартних інструментів веб-розробки Java і стандартних API. Додаток взаємодіє з середовищем за допомогою стандарту Java Servlet і може використовувати стандартні технології веб-додатків, такі як сторінки JavaServer Pages (JSP).

Середовище виконання Java використовує Java 6. SDK Java App Engine підтримує розробку додатків за допомогою Java 5 і 6.

Середовище включає платформу Java SE Runtime Environment (JRE) 6 і бібліотеки. Обмеження на тестове середовище реалізовані в JVM. Додаток може використовувати байтів код JVM або бібліотеки в межах обмежень тестового середовища. Наприклад, при спробі байтового коду відкрити сонет або записати файл виникне виключення середовища виконання.

Доступ до більшості служб App Engine можна отримати через стандартні API Java. Для сховища даних App Engine SDK Java містить реалізації інтерфейсів об'єктів даних Java (JDO) і Java Persistence API (JPA). Щоб відправляти повідомлення по електронній пошті за допомогою служби Mail App Engine, можна використовувати APIJavaMail. У API HTTP java.net є доступ до служби отримання даних по URL App Engine. Крім того, для своїх

служб App Engine включає низькорівневі API, які реалізують додаткові адаптери і дозволяють використовувати служби безпосередньо з програми. Ознайомтеся з документацією по API сховищ даних, кеша пам'яті, отримання даних поURL, пошти, зображень та облікових записів Google.

Зазвичай, щоб розробити веб-додатки для JVM, Java-розробники використовують мову програмування Java і API. Використовуючи сумісні з JVM компілятори та інтерпретатори, можна розробляти веб-додатки на інших мовах, таких як JavaScript, Ruby і Scala.

Завдяки середовищу виконання Python App Engine можна створювати додатки за допомогою мови програмування Python і виконувати їх за допомогою оптимізованого інтерпретатора Python. App Engine включає різноманітні API та інструменти для розробки веб-додатків Python, в тому числі API моделювання збагачених даних, просту у використанні інфраструктуру веб-додатків і інструменти для управління і доступу до даних додатка. Для розробки веб-додатків Python можна скористатися перевагами широкого набору бібліотек і інфраструктур, наприклад Django.

Середа виконання Python використовує Python версії 2.5.2. Для майбутнього випуску розглядається можливість підтримки Python 3.

Середа Python містить стандартну бібліотеку Python. Зрозуміло, що не всі функції бібліотеки можна використовувати в тестовому середовищі. Наприклад, при виклику методу, що відкриває сокет або записуючого в файл, виникне виключення. Для зручності відключені кілька модулів стандартної бібліотеки, ключові функції яких не підтримуються середовищем виконання. Виконання імпортуючого їх коду призводить до помилки.

Код програми, створений для середовища Python, повинен бути написаний виключно на Python. Розширення, написані на мові C, – не підтримуються.

Середовище Python надає потужні API Python для служб сховища даних, облікових записів Google, отримання URL та електронної пошти. App Engine також надає просту інфраструктуру веб-додатку Python під назвою webapp, яка полегшує створення додатків.

Разом з додатком можна завантажувати сторонні бібліотеки, але вони повинні бути реалізовані на чистому Python і не повинні вимагати непідтримуваних модулів стандартної бібліотеки.

App Engine надає потужну службу розподіленого зберігання даних, що включає механізм запитів і транзакції. Розширення розподіленої бази даних за рахунок даних аналогічно розширенню розподіленого веб-сервера за рахунок трафіку.

Сховище даних App Engine не схоже на звичайну реляційну базу даних. Об'єкти даних, або "записи", мають вигляд і володіють набором властивостей. За допомогою запитів можна отримувати записи певного виду, відфільтровані і відсортовані за значеннями властивостей. Значення властивостей можуть бути будь-якими з підтримуваних типів значень властивостей.

Для об'єктів сховища даних не потрібна схема. Структура об'єктів даних визначається в коді програми. Інтерфейси JDO / JPA Java і сховища даних

Python включають функції для застосування структури в додатку. Додаток може отримати прямий доступ до сховища даних, щоб реалізувати потрібну частину структури.

Сховище даних узгоджено і використовує оптимістичне управління паралельними транзакціями. Оновлення запису відбувається в транзакції, яка виконується повторно певну кількість разів, якщо інші процеси одночасно намагаються оновити той ж запис. Додаток може виконувати кілька операцій зі сховищем даних в одній транзакції. Ці операції або всі будуть успішні, або всі будуть невдалі, що забезпечує цілісність даних.

Сховище даних реалізує транзакції у своїй розподіленої мережі за допомогою "груп записів". Транзакція здійснює дії над записами в одній групі. Записи кожної з груп зберігаються разом для ефективного виконання транзакцій. При створенні записів додаток може приєднувати їх до груп.

App Engine підтримує інтеграцію програми з акаунтами Google для аутентифікації користувачів. Ваш додаток може дозволити користувачеві увійти в свій аккаунт Google і отримати доступ до адреси електронної пошти і псевдонімам, пов'язаним з членством. Використання облікових записів Google дає користувачеві можливість швидше почати використовувати ваш додаток, оскільки йому не доведеться створювати новий акаунт. Це також знімає з вас необхідність реалізовувати систему акаунтів користувачів тільки для свого додатку.

Якщо програма працює в Службах Google, вона може використовувати ті ж функції для учасників вашої організації і акаунтів Служб Google.

API користувачів може також сказати додаткам, чи є поточний користувач зареєстрованим адміністратором додатку. Це спрощує реалізацію адміністративних зон сайту.

App Engine надає набір служб, що дозволяють виконувати рядові операції при управлінні додатком. Для доступу до цих служб надані наступні API.

Додатки можуть отримувати доступ до ресурсів в Інтернеті, наприклад до веб-служб або інших даних, за допомогою служби отримання URL App Engine. Служба отримання даних по URL забезпечує отримання веб-ресурсів за допомогою тієї ж високошвидкісної інфраструктури Google, яка отримує веб-сторінки для багатьох інших продуктів Google.

– Електронна пошта.

Додатки можуть відправляти повідомлення електронної пошти за допомогою поштової служби App Engine. Для відправлення електронних повідомлень ця служба використовує інфраструктуру Google.

– Memcache.

Служба Memcache надає вашому застосуванню високопродуктивний кеш пам'яті, що використовує структуру ключ-значення, до якого може отримувати доступ кілька примірників додатка. Кеш пам'яті в нагоді для даних, що не вимагають постійного зберігання та функцій роботи з транзакціями, які надає сховище даних, наприклад для тимчасових даних або даних, що копіюються зі сховища в кеш для прискорення доступу.

– Робота з зображеннями.

Служба зображень дозволяє додатку працювати із зображеннями. За допомогою цього API можна змінювати розмір, обрізати, повертати і відображати зображення в форматах JPEG і PNG.

– Заплановані завдання.

Служба Cron дозволяє планувати завдання для виконання через певні інтервали. Докладніше про неї можна дізнатися в документації по службі Cron Python і Java.

– Процес розробки.

Інструментарій розробки App Engine (SDK) для Java і Python включає додаток на веб-сервері, який імітує служби App Engine на локальному комп'ютері. Кожен SDK включає всі API і бібліотеки, що перебувають у App Engine. Крім того, веб-сервер імітує безпечне тестове середовище, що включає перевірку на доступ до системних ресурсів, заборонений в App Engine.

Кожен SDK також включає інструмент для додавання додатку в App Engine. Після створення коду програми, статичних файлів і файлів конфігурації запустіть цей інструмент, щоб завантажити дані. Інструмент запросить адресу електронної пошти та пароль вашого облікового запису Google.

При створенні нового випуску програми, яка вже працює в App Engine, ви зможете завантажити його як нову версію. Стара версія буде працювати для користувачів доти, поки ви не перейдете на нову. Ви можете тестувати нову версію в App Engine, поки працює стара.

SDK Java виконується на будь-якій платформі з Java 5 або Java 6. SDK доступний у вигляді ZIP-файлу. При використанні середовища розробки Eclipse, щоб створити, перевірити і додати програми App Engine, можна використовувати плагін Google для Eclipse. SDK також містить інструменти, що працюють з командного рядка, які дозволяють запускати сервер розробки та додавати додатки.

SDK Python реалізований на чистому Python і виконується на будь-якій платформі з Python 2.5, у тому числі Windows, Mac OS X і Linux. SDK доступний у вигляді Zip-файлу, а для Windows і Mac OS X доступні програми установки.

Консоль адміністрування - це веб-інтерфейс для управління додатками, що працюють в App Engine. Її можна використовувати для створення нових додатків, налаштування доменних імен, зміни робочої версії програми, вивчення доступу і журналів помилок і перегляду сховища даних програми.

– Квоти і обмеження.

Створити додаток в App Engine не тільки просто, але й безкоштовно! Ви можете створити його і опублікувати додаток, який можна буде використовувати відразу ж, безкоштовно і без додаткових вимог. Додаток з безкоштовним акаунтом може використовувати до 500 МБ сховища даних і до п'яти мільйонів переглядів сторінок на місяць. Якщо потрібно більше, включіть оплату, встановіть максимальний денний бюджет і розподіліть його між ресурсами у відповідності зі своїми потребами.

Для аккаунта розробника можна зареєструвати до 10 додатків.

Кожному додатку надаються ресурси в межах обмежень або "квот". Квота визначає обсяг певного ресурсу, який можна використовувати протягом календарного дня. Найближчим часом буде можливо налаштувати деякі з цих квот, оплативши додаткові ресурси.

Для деяких функцій обмеження не пов'язані з квотами, а призначені для збереження стабільності системи. Наприклад, якщо додаток викликається для виконання веб-запиту, воно має створити відповідь протягом 30 секунд. Якщо цей процес триває занадто довго, то він припиняється, а сервер повертає користувачеві код помилки. Таймаут запиту динамічний і може зменшуватися для економії ресурсів, якщо обробник запитів досягає його занадто часто.

Інший приклад обмеження обслуговування - кількість повернених запитом результатів. Запит може повернути не більше 1000 результатів. Запити, які могли б повернути більше, повертають тільки максимально допустиму кількість. У цьому випадку такий запит швидше за все не поверне результати до настання таймаута, але завдяки обмеженню ресурси сховища даних будуть зекономлені.

Спроби обійти або перевищити квоти, наприклад, виконуючи програми в декількох спільно працюючих акаунтах, порушують *Умови надання послуг* і можуть призвести до відключення застосування чи закриття акаунтів.

Список квот і пояснення системи квот, включаючи квоти, які можна збільшити, включивши оплату, можна подивитися в статті *Квоти*.

Література:

1. http://hostinfo.ru
2. http://www.google.com/apps

ЛЕКЦІЯ 11. КОНСОЛІДАЦІЯ, ВІРТУАЛІЗАЦІЯ ІТ-ІНФРАСТРУКТУРИ. ВІРТУАЛІЗАЦІЯ ЗАСТОСУВАНЬ (ДОДАТКІВ)

Інформаційні технології принесли в життя сучасного суспільства безліч корисних і цікавих речей. Кожен день винахідливі і талановиті люди придумують все нові і нові застосування комп'ютерів як ефективних інструментів виробництва, розваг та співробітництва. Безліч різних програмних і апаратних засобів, технологій і сервісів дозволяють нам щодня підвищувати зручність і швидкість роботи з інформацією. Усе складніше і складніше виділити з падаючого на нас потоку технологій дійсно корисні і навчитися застосовувати їх з максимальною користю. У цій лекції йтиметься про ще одну неймовірно перспективну і по – справжньому ефективну технологію,яка стрімко вривається у світ комп'ютерів – технологію віртуалізацію, яка займає ключове місце в концепції «хмарних» обчислень.

11.1. Технології віртуалізації

Згідно зі статистикою середній рівень завантаження процесорних потужностей у серверів під управлінням Windows не перевищує 10%, у Unix – систем цей показник кращий, проте в середньому не перевищує 20 %. Низька ефективність використання серверів пояснюється широко застосовуваним з початку 90 – х років підходом " один додаток – один сервер", тобто кожен раз для розгортання нової програми компанія набуває новий сервер. Очевидно, що на практиці це означає швидке збільшення серверного парку і як наслідок – зростання витрат на його адміністрування, енергоспоживання та охолодження, а також потреба в додаткових приміщеннях для установки все нових серверів і придбанні ліцензій на серверну ОС.

Віртуалізація ресурсів фізичного сервера дозволяє гнучко розподіляти їх між додатками, кожен з яких при цьому "бачить" тільки призначені йому ресурси і " вважає", що йому виділено окремий сервер, тобто в даному випадку реалізується підхід " один сервер – кілька додатків ", але без зниження продуктивності, доступності та безпеки серверних додатків. Крім того, рішення віртуалізації дають можливість запускати в розділах різні ОС за допомогою емуляції їх системних викликів до апаратних ресурсів сервера.

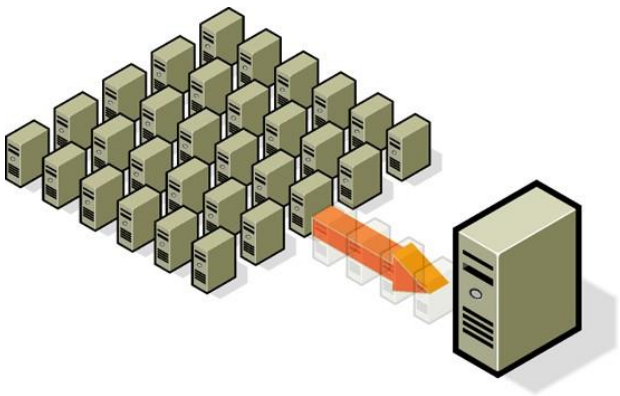


Рисунок 11.1. – Віртуалізація має на увазі запуск на одному фізичному комп'ютері декількох віртуальних комп'ютерів

В основі віртуалізації лежить можливість одного комп'ютера виконувати роботу декількох комп'ютерів завдяки розподілу його ресурсів на декілька середовищ. За допомогою віртуальних серверів і віртуальних настільних комп'ютерів можна розмістити кілька ОС і кілька додатків в єдиному розташування. Таким чином, фізичні та географічні обмеження перестають мати якесь значення. Крім енергозбереження та скорочення витрат завдяки більш ефективному використанню апаратних ресурсів, віртуальна інфраструктура забезпечує високий рівень доступності ресурсів, більш ефективну систему управління, підвищену безпеку і вдосконалену систему відновлення в критичних ситуаціях.

У широкому розумінні поняття віртуалізації являє собою приховування справжньої реалізації якогось процесу або об'єкта від істинного його подання для того, хто ним користується. Продуктом віртуалізації є щось зручне для використання, насправді, маючи більш складну або зовсім іншу структуру, відмінну від тієї, яка сприймається при роботі з об'єктом. Іншими словами, відбувається відділення представлення від реалізації чого -небудь. Віртуалізація покликана абстрагувати програмне забезпечення від апаратної частини.

У комп'ютерних технологіях під терміном «віртуалізація» зазвичай розуміється абстракція обчислювальних ресурсів і надання користувачеві системи, яка «інкапсулює» (приховує в собі) власну реалізацію. Простіше кажучи, користувач працює з зручним для себе представленням об'єкта, і для нього не має значення, як об'єкт влаштований в дійсності.

Зараз можливість запуску декількох віртуальних машин на одній фізичній викликає великий інтерес серед комп'ютерних фахівців, не тільки тому, що це підвищує гнучкість IT – інфраструктури, але й тому, що віртуалізація, насправді, дозволяє економити гроші.

Історія розвитку технологій віртуалізації налічує понад сорок років. Компанія IBM була першою, хто задумався про створення віртуальних середовищ для різних призначених для користувача завдань (ще в мейнфреймах). У 60 – х роках минулого століття віртуалізація представляла чисто науковий інтерес і була оригінальним рішенням для ізоляції комп'ютерних систем в рамках одного фізичного комп'ютера. Після появи персональних комп'ютерів інтерес до віртуалізації дещо послабився зважаючи на бурхливий розвиток операційних систем, які потребували адекватні вимоги до апаратного забезпечення того часу. Однак бурхливе зростання апаратних потужностей комп'ютерів наприкінці дев'яностих років минулого століття змусило IT – спільноту знову згадати про технології віртуалізації програмних платформ.

У 1999 р. компанія VMware представила технологію віртуалізації систем на базі x86 в якості ефективного засобу, здатного перетворити системи на базі x86 в єдину апаратну інфраструктуру загального користування та призначення, що забезпечує повну ізоляцію, мобільність і широкий вибір ОС для прикладних середовищ. Компанія VMware була однією з перших, хто зробив серйозну ставку виключно на віртуалізацію. Як показав час, це виявилося абсолютно

виправданим. Сьогодні WMware пропонує комплексну віртуалізаційну платформу четвертого покоління VMware vSphere 4, яка включає засоби як для окремого ПК, так і для центру обробки даних. Ключовим компонентом цього програмного комплексу є гіпервізор VMware ESX Server. Пізніше в «битву» за місце у цьому модному напрямку розвитку інформаційних технологій включилися такі компанії як Parallels (раніше SWsoft), Oracle (Sun Microsystems), Citrix Systems (XenSourse).

Корпорація Microsoft вийшла на ринок засобів віртуалізації в 2003 р. з придбанням компанії Connectix, випустивши свій перший продукт Virtual PC для настільних ПК. З тих пір вона послідовно нарощувала спектр пропозицій у цій галузі і на сьогодні майже завершила формування віртуалізаційних платформ, до складу якох входять такі рішення як Windows 2008 Server R2 с компонентом Hyper -V, Microsoft Application Virtualization (App – v), Microsoft Virtual Desktop Infrastructure (VDI), Remote Desktop Services, System Center Virtual Machine Manager.

На сьогоднішній день постачальники технологій віртуалізації пропонують надійні і легкокеровані платформи, а ринок цих технологій переживає справжній бум. За оцінками провідних експертів, зараз віртуалізація входить до трійки найбільш перспективних комп'ютерних технологій. Багато експертів пророкують, що до 2015 року близько половини всіх комп'ютерних систем будуть віртуальними.

Підвищений інтерес до технологій віртуалізації в даний час невипадковий. Обчислювальна потужність нинішніх процесорів швидко зростає, і питання навіть не в тому, на що цю потужність витрачати, а в тому, що сучасна «мода» на двоядерні і багатоядерні системи, що проникла вже і в персональні комп'ютери (ноутбуки та десктопи), як не можна краще дозволяє реалізувати багатющий потенціал ідей віртуалізації операційних систем та програм, виводячи зручність користування комп'ютером на новий якісний рівень. Технології віртуалізації стають одним з ключових компонентів (у тому числі, і маркетингових) в найновіших і майбутніх процесорах Intel і AMD, в операційних системах від Microsoft та ряду інших компаній.

Наведемо основні переваги технологій віртуалізації:

1. Ефективне використання обчислювальних ресурсів. Замість 3х, а то і 10 серверів, завантажених на 5 -20 % можна використовувати один, використовуваний на 50 -70 %. Крім іншого, це ще й економія електроенергії, а також значне скорочення фінансових вкладень: придбається один високотехнологічний сервер, що виконує функції 5 -10 серверів. За допомогою віртуалізації можна досягти значно більш ефективного використання ресурсів, оскільки вона забезпечує об'єднання стандартних ресурсів інфраструктури в єдиний пул і долає обмеження застарілої моделі «одно додатків на сервер».

2. Скорочення витрат на інфраструктуру: Віртуалізація дозволяє скоротити кількість серверів і пов'язаного з ними IT-обладнання в інформаційному центрі. У результаті цього потреби в обслуговуванні, електроживленні й охолодженні матеріальних ресурсів скорочуються, і на IT витрачається значно менше коштів.

3. Зниження витрат на програмне забезпечення. Деякі виробники програмного забезпечення ввели окремі схеми ліцензування спеціально для віртуальних середовищ. Так, наприклад, купуючи одну ліцензію на Microsoft Windows Server 2008 Enterprise, ви отримуєте право одночасно її використовувати на 1 фізичному сервері і 4 віртуальних (в межах одного сервера), а Windows Server 2008 Datacenter ліцензується тільки на кількість процесорів і може використовуватися одночасно на необмеженій кількості віртуальних серверів.

4. Підвищення гнучкості і швидкості реагування системи: Віртуалізація пропонує новий метод управління IT-інфраструктурою і допомагає IT – адміністраторам затрачати менше часу на виконання повторюваних завдань – наприклад, на ініціацію, налаштування, відстеження і технічне обслуговування. Багатосистемні адміністратори відчували неприємності, коли «валиться» сервер. І не можна, витягнувши жорсткий диск, переставивши його в інший сервер, запустити все як раніше... А установка? пошук драйверів, настройка, запуск... і на все потрібні час і ресурси. При використанні віртуального сервера – можливий моментальний запуск на будь - якому “залізі“, а якщо немає подібного сервера, то можна скачати готову віртуальну машину з встановленим та настроєним сервером, з бібліотек, підтримуваних компаніями розробниками гіпервізорів (програм для віртуалізації).

5. Несумісні додатки можуть працювати на одному комп'ютері. При використанні віртуалізації на одному сервері можлива установка linux і windows серверів, шлюзів, баз даних і інших абсолютно несумісних в рамках однієї не віртуалізованої системи додатків.

6. Підвищення доступності додатків і забезпечення безперервності роботи підприємства: Завдяки надійній системі резервного копіювання та міграції віртуальних середовищ цілком без перерв в обслуговуванні ви зможете скоротити періоди планового простою і забезпечити швидке відновлення системи в критичних ситуаціях. "Падіння» одного віртуального сервера не веде до втрати інших віртуальних серверів. Крім того, у разі відмови одного фізичного сервера можливо зробити автоматичну заміну на резервний сервер. Причому це відбувається не помітно для користувачів без перезагрузки. Тим самим забезпечується безперервність бізнесу.

7. Можливості легкої архівації. Оскільки жорсткий диск віртуальної машини зазвичай представляється у вигляді файлу певного формату, розташований на якому -небудь фізичному носії, віртуалізація дає можливість простого копіювання цього файлу на резервний носій як засіб архівування і резервного копіювання всієї віртуальної машини цілком. Можливість підняти з архіву сервер повністю – ще одна чудова особливість. Можливо підняти сервер з архіву, не знищуючи поточний сервер і подивитися стан справ за минулий період.

8. Підвищення керованості інфраструктури: використання централізованого управління віртуальною інфраструктурою дозволяє скоротити

час на адміністрування серверів, забезпечує балансування навантаження і "живу" міграцію віртуальних машин.

Віртуальною машиною будемо називати програмне або апаратне середовище, яка приховує справжню реалізацію якогось процесу або об'єкту від його справжнього подання.

Віртуальна машина – це повністю ізольований програмний контейнер, який працює з власною ОС і додатками, подібно фізичному комп'ютеру. Віртуальна машина діє так само, як фізичний комп'ютер, і містить власні віртуальні (тобто програмні) ОЗУ, жорсткий диск і мережевий адаптер.

ОС не може розрізнити віртуальну і фізичну машини. Те ж саме можна сказати про додатки та інших комп'ютерах в мережі. Навіть сама віртуальна машина вважає себе «справжнім» комп'ютером. Але незважаючи на це віртуальні машини складаються виключно з програмних компонентів і не включають обладнання. Це дає їм низку унікальних переваг над фізичною обладнанням.

Рисунок 11.2. – Віртуальна машина

Розглянемо основні особливості віртуальних машин більш детально:

1. Сумісність. Віртуальні машини, як правило, сумісні з усіма стандартними комп'ютерами. Як і фізичний комп'ютер, віртуальна машина працює під управлінням власної гостьовий оперативної системи і виконує власні додатки. Вона також містить всі компоненти, стандартні для фізичного комп'ютера (материнську плату, відеокарту, мережевий контролер і т.д.). Тому віртуальні машини повністю сумісні з усіма стандартними операційними системами, додатками і драйверами пристроїв. Віртуальну машину можна використовувати для виконання будь -якого програмного забезпечення, придатного для відповідного фізичного комп'ютера.

2. Ізольованість. Віртуальні машини повністю ізольовані один від одного, так ніби вони є фізичними комп'ютерами Віртуальні машини можуть використовувати загальні фізичні ресурси одного комп'ютера і при цьому залишатися повністю ізольованими один від одного, як якщо б вони були окремими фізичними машинами. Наприклад, якщо на одному фізичному сервері запущено чотири віртуальних машини, і одна з них дає збій, це не впливає на доступність решти трьох машин. Ізольованість – важлива причина набагато більш високої доступності і безпеки програм, виконуваних у

віртуальному середовищі, в порівнянні з додатками, виконуваними в стандартній, невіртуалізірованій системі.

3. Інкапсуляція. Віртуальні машини повністю інкапсулюють обчислювальне середовище. Віртуальна машина являє собою програмний контейнер, зв'язуючий, або «інкапсулюючий» повний комплект віртуальних апаратних ресурсів, а також ОС і всі її додатки в програмному пакеті. Завдяки інкапсуляції віртуальні машини стають неймовірно мобільними і зручними в управлінні. Наприклад, віртуальну машину можна перемістити або скопіювати з одного пункту до іншого так само, як будь -який інший програмний файл. Крім того, віртуальну машину можна зберегти на будь -якому стандартному носії даних: від компактної карти Flash -пам'яті USB до корпоративних мереж зберігання даних.

4. Незалежність від обладнання. Віртуальні машини повністю незалежні від базового фізичного обладнання, на якому вони працюють. Наприклад, для віртуальної машини з віртуальними компонентами (ЦП, мережевою картою, контролером SCSI) можна задати налаштування, абсолютно не збігаються з фізичними характеристиками базового апаратного забезпечення. Віртуальні машини можуть навіть виконувати різні операційні системи (Windows, Linux та ін) на одному і тому ж фізичному сервері. У поєднанні з властивостями інкапсуляції і сумісності, апаратна незалежність забезпечує можливість вільно переміщувати віртуальні машини з одного комп'ютера на базі x86 на інший, не змінюючи драйвери пристроїв, ОС або програми. Незалежність від обладнання також дає можливість запускати в поєднанні абсолютно різні ОС і додатки на одному фізичному комп'ютері.

Розглянемо основні різновиди віртуалізації, такі як:

- віртуалізація серверів (повна віртуалізація і паравіртуалізації);
- віртуалізація на рівні операційних систем;
- віртуалізація додатків;
- віртуалізація уявлень.

11.2. Віртуалізація застосувань (додатків)

Віртуалізація додатків має на увазі застосування моделі сильної ізоляції прикладних програм з керованою взаємодією з ОС, при якій віртуалізується кожен екземпляр додатків, всі його основні компоненти: файли (включаючи системні), реєстр, шрифти, INI – файли, COM – об'єкти, служби. Додаток виповнюється без процедури інсталяції в традиційному її розумінні і може запускатися прямо з зовнішніх носіїв (наприклад, з флеш – карт або з мережевих папок).

Рисунок 11.3. – Віртуалізація додатків

З точки зору IT – відділу, такий підхід має очевидні переваги: прискорення розгортання настільних систем і можливість управління ними, зведення до мінімуму не тільки конфліктів між додатками, а й потреби у тестуванні додатків на сумісність. Дана технологія дозволяє використовувати на одному комп'ютері, а точніше в одній і тій же операційній системі кілька несумісних між собою додатків одночасно. Віртуалізація додатків дозволяє користувачам запускати одне і теж заздалегідь сконфігурованій додаток або групу додатків з сервера. При цьому додатки будуть працювати незалежно один від одного, не вносячи жодних змін в операційну систему. Фактично саме такий варіант віртуалізації використовується в Sun Java Virtual Machine, Microsoft Application Virtualization (раніше називався Softgrid), Thinstall (на початку 2008 р. увійшла до складу VMware), Symantec / Altiris.

ЛЕКЦІЯ 12. ВІРТУАЛІЗАЦІЯ РОБОЧИХ МІСЦЬ

Віртуалізація уявлень (робочих місць) Віртуалізація уявлень має на увазі емуляцію інтерфейсу користувача. Тобто користувач бачить додаток і працює з ним на своєму терміналі, хоча насправді додаток виконується на віддаленому сервері, а користувачеві передається лише картинка віддаленої програми. Залежно від режиму роботи користувач може побачити віддалений робочий стіл і запущене на ньому додаток, або тільки саме вікно програми.

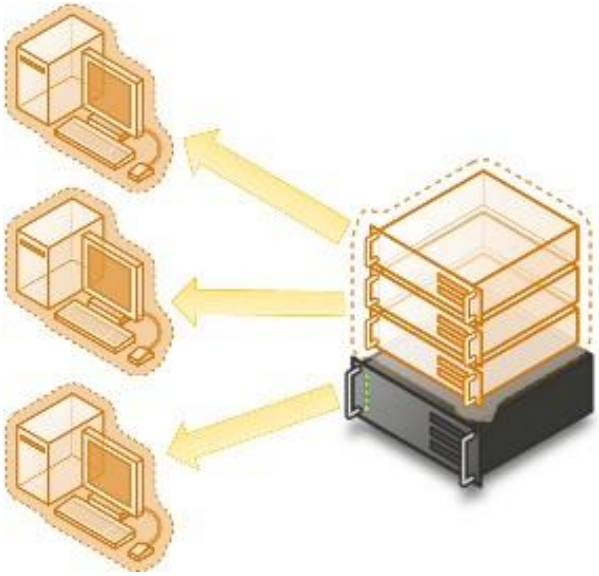


Рисунок 12.1. – Віртуалізація уявлень

Потреби бізнесу змінюють наше уявлення про організацію робочого процесу. Персональний комп'ютер, який став за останні десятиліття невід'ємним атрибутом офісу і засобом виконання більшості офісних завдань, перестає встигати за зростаючими потребами бізнесу. Реальним інструментом користувача виявляється програмне забезпечення, яке лише прив'язане до ПК, роблячи його посередником корпоративної інформаційної системи. В результаті активний розвиток отримують «хмарні» обчислення, коли користувачі мають доступ до власних даних, але не управляють і не замислюються про інфраструктуру, операційну систему і власне програмне забезпечення, з яким вони працюють.

Разом з тим, із зростанням масштабів організацій, використання в IT – інфраструктурі користувацьких ПК викликає ряд складнощів:

– великі операційні витрати на підтримку комп'ютерного парку;

– складність, пов'язана з управлінням настільними ПК ;

– забезпечення користувачам безпечного і надійного доступу до ПЗ і додаткам необхідним для роботи ;

– технічний супровід користувачів;

– встановлення та оновлення ліцензій на ПЗ і технічне обслуговування;

– резервне копіювання і т.д.

Втікти від цих складнощів і скоротити витрати, пов'язані з їх вирішенням, можливо завдяки застосуванню технології віртуалізації робочих місць співробітників на базі інфраструктури віртуальних ПК – Virtual Desktop Infrastructure (VDI). VDI дозволяє відокремити користувальницьке ПЗ від апаратної частини – персонального комп'ютера, – і здійснювати доступ до клієнтських додатків через термінальні пристрої.

VDI – комбінація сполук з віддаленим робочим столом і віртуалізацією. На обслуговуючих серверах працює безліч віртуальних машин, з такими клієнтськими операційними системами, як Windows 7, Windows Vista і Windows XP або Linux операційними системами. Користувачі дистанційно підключаються до віртуальної машини свого робочого середовища.

VDI повністю ізолює віртуальне середовище користувачів від інших віртуальних середовищ, так як кожен користувач підключається до окремої віртуальної машини. Іноді використовується статична інфраструктура VDI, в якій користувач завжди підключається до тієї ж віртуальної машині, в інших випадках динамічна VDI, в якій користувачі динамічно підключаються до різних віртуальних машин, і віртуальні машини створюються в міру необхідності. При використанні будь -якої моделі важливо зберігати дані користувачів поза віртуальних машин і швидко надавати додатки.

Поряд з централізованим управлінням і простим наданням комп'ютерів, VDI забезпечує доступ до робочого середовища з будь -якого місця, якщо користувач може дистанційно підключитися до сервера.

Уявімо, що на клієнтському комп'ютері виникла неполадка. Доведеться виконати діагностику і, можливо, перевстановити операційну систему. Завдяки VDI в разі неполадок можна просто видалити віртуальну машину і за кілька секунд створити нове середовище, за допомогою створеного заздалегідь шаблону віртуальної машини. VDI забезпечує додаткову безпеку, так як дані не зберігаються локально на настільному комп'ютері або ноутбуці.

Як приклад віртуалізації уявлень можна розглядати і технологію тонких терміналів, які фактично віртуалізують робочі місця користувачів настільних систем: користувач не прив'язаний до якогось конкретного ПК, а може отримати доступ до своїх файлів і додатків, які розташовуються на сервері, з будь -якого віддаленого терміналу після виконання процедури авторизації. Всі команди користувача і зображення сеансу на моніторі емулюються за допомогою ПЗ керування тонкими клієнтами. Застосування цієї технології дозволяє централізувати обслуговування клієнтських робочих місць і різко скоротити витрати на їх підтримку – наприклад, для переходу на наступну версію клієнтської програми нове ПЗ потрібно інсталювати тільки один раз на сервері.

Рисунок 21.2. – Приклад тонкого клієнта. Термінал Sun Ray

Одним з найбільш відомих тонких клієнтів є термінал Sun Ray, для організації роботи якого використовується програмне забезпечення Sun Ray Server Software. Для початку сеансу Sun Ray достатньо лише вставити в цей пристрій ідентифікаційну смарт -карту. Застосування смарт – картки істотно підвищує мобільність користувача – він може переходити з одного Sun Ray на інший, переставляючи між ними свою картку і відразу продовжувати роботу зі своїми додатками з того місця, де він зупинився на попередньому терміналі. А відмова від жорсткого диска не тільки забезпечує мобільність користувачів і підвищує безпеку даних, але й істотно знижує енергоспоживання в порівнянні з звичайними ПК, тому термінал ВС не має вентилятора і працює практично безшумно. Крім того, скорочення числа компонентів тонкого терміналу зменшує і ризик виходу його з ладу, а отже, економить витрати на його обслуговування. Ще одна перевага Sun Ray – це істотно розширений в порівнянні із звичайними ПК життєвий цикл продукту, оскільки в ньому немає компонентів, які можуть морально застаріти.

ЛЕКЦІЯ 13. ВІРТУАЛІЗАЦІЯ СЕРВЕРІВ

Сьогодні, говорячи про технології віртуалізації, як правило, мають на увазі віртуалізацію серверів, так як остання стає найбільш популярним рішенням на ринку IT. Віртуалізація серверів має на увазі запуск на одному фізичному сервері декількох віртуальних серверів. Віртуальні машини або сервера являють собою програми, запущені на хостовій операційній системі, які емулюють фізичні пристрої сервера. На кожній віртуальній машині може бути встановлена операційна система, на яку можуть бути встановлені додатки і служби. Типові представники це продукти VmWare (ESX, Server, Workstation) і Microsoft (Hyper -V, Virtual Serer, Virtual PC).

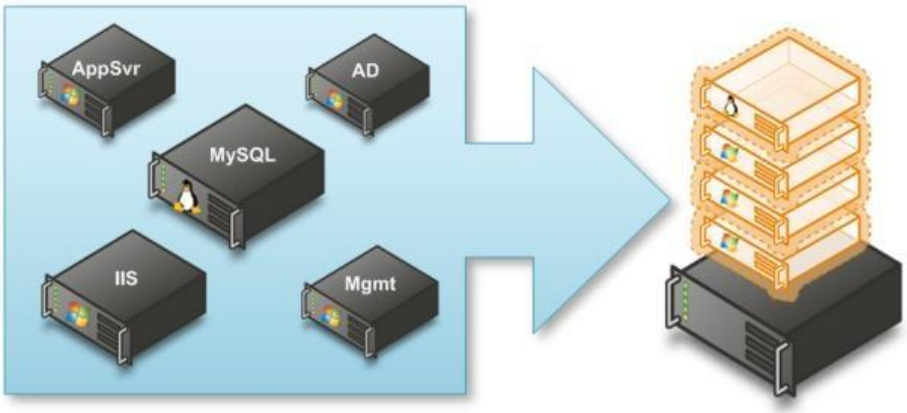


Рисунок 13.1. – Віртуалізація серверів

Центри обробки даних використовують великий простір і величезну кількість енергії, особливо якщо додати до цього супроводжуючі їх системи охолодження та інфраструктуру. Засобами технологій віртуалізації виконується консолідація серверів, розташованих на великій кількості фізичних серверів у вигляді віртуальних машин на одному високопродуктивному сервері.

Число фізичних машин, необхідних для роботи в якості серверів зменшується, що знижує кількість енергії, необхідної для роботи машин і простір, необхідний для їх розміщення. Скорочення в кількості серверів і просторі зменшує кількість енергії, необхідної для їх охолодження. При меншій витраті енергії виробляється менша кількість вуглекислого газу. Даний показник, наприклад в Європі, має досить важливу роль.

Важливим чинником є фінансова сторона. Віртуалізація є важливим моментом економії. Віртуалізація не тільки зменшує потребу в придбанні додаткових фізичних серверів, але і мінімізує вимоги до їх розміщення. Використання віртуального сервера надає переваги по швидкості впровадження, використання та управління, що дозволяє зменшити час очікування розгортання якого проекту.

Не так давно з'явилися моделі останнього покоління процесорів в архітектурі x86 корпорацій AMD і Intel, де виробники вперше додали технології апаратної підтримки віртуалізації. До цього віртуалізація підтримувалася

програмно, що природно приводила до великих накладних витрат продуктивності.

Для персональних комп'ютерів вісімдесятих років двадцятого століття – проблема віртуалізації апаратних ресурсів, здавалося б, не існувала за визначенням, оскільки кожен користувач отримував в своє розпорядження весь комп'ютер зі своєю ОС. Але у міру зростання потужності ПК і розширення сфери застосування x86 – систем ситуація швидко змінилася. "Діалектична спіраль" розвитку зробила свій черговий виток, і на рубежі століть розпочався черговий цикл посилення доцентрових сил по концентрації обчислювальних ресурсів. На початку нинішнього десятиліття на тлі зростаючої зацікавленості підприємств у підвищенні ефективності своїх комп'ютерних засобів стартував новий етап розвитку технологій віртуалізації, який зараз переважно пов'язується саме з використанням архітектури x86.

Зазначимо, що хоча в ідеях x86 – віртуалізації в теоретичному плані начебто нічого невідомого раніше не було, йшлося про якісно новий для IT – галузі явищі в порівнянні з ситуацією 20 – річної давності. Справа в тому, що в апаратно – програмній архітектурі мейнфреймів і Unix – комп'ютерів питання віртуалізації відразу вирішувалися на базовому рівні апаратному рівні. Система ж x86 будувалася зовсім не з розрахунку на роботу в режимі датацентрів, і її розвиток у напрямку віртуалізації – це досить складний еволюційний процес з безліччю різних варіантів вирішення завдання.

Важливий момент полягає також в якісно різних бізнес – моделях розвитку мейнфреймів і x86. У першому випадку мова йде фактично про моновендорном програмно – апаратному комплексі для підтримки досить обмеженого кола прикладного ПЗ для досить вузького кола великих замовників. У другому – ми маємо справу з децентралізованим співтовариством виробників техніки, постачальників базового ПЗ і величезною армією розробників прикладного програмного забезпечення.

Використання коштів x86 – віртуалізації почалося наприкінці 90 – х з робочих станцій: одночасно із збільшенням кількості версій клієнтських ОС постійно зростала і кількість людей (розробників ПЗ, фахівців з технічної підтримки, експертів), яким потрібно було на одному ПК мати відразу кілька копій різних ОС.

Віртуалізація для серверної інфраструктури стала застосовуватися трохи пізніше, і пов'язано це було, перш за все, з вирішенням завдань консолідації обчислювальних ресурсів. Але тут відразу сформувалося два незалежних напрямки:

– підтримка неоднорідних операційних середовищ (в тому числі, для роботи успадкованих додатків). Цей випадок найбільш часто зустрічається в рамках корпоративних інформаційних систем. Технічно проблема вирішується шляхом одночасної роботи на одному комп'ютері декількох віртуальних машин, кожна з яких включає примірник операційної системи. Але реалізація цього режиму виконувалася за допомогою двох принципово різних підходів: повної віртуалізації і паравіртуалізації;

– підтримка однорідних обчислювальних середовищ увазі ізоляцію служб в рамках одного примірника ядра операційної системи (віртуалізація на рівні ОС), що найбільш характерно для хостингу додатків провайдерами послуг. Звичайно, тут можна використовувати і варіант віртуальних машин, але набагато ефективніше створення ізольованих контейнерів на базі одного ядра однієї ОС.

Наступний життєвий етап технологій x86 – віртуалізації стартував у 2004 – 2006 рр.. і був пов'язаний з початком їх масового застосування в корпоративних системах. Відповідно, якщо раніше розробники в основному займалися створенням технологій виконання віртуальних середовищ, то тепер на перший план стали виходити завдання управління цими рішеннями та їх інтеграції в загальну корпоративну IT – інфраструктуру. Одночасно позначилося помітне підвищення попиту на віртуалізацію з боку персональних користувачів (але якщо в 90 – х це були розробники і тестери, то зараз мова вже йде про кінцевих користувачів як професійних, так і домашніх).

Багато труднощів і проблем розробки технологій віртуалізації пов'язані з подоланням успадкованих особливостей програмно – апаратної архітектури x86. Для цього існує кілька базових методів:

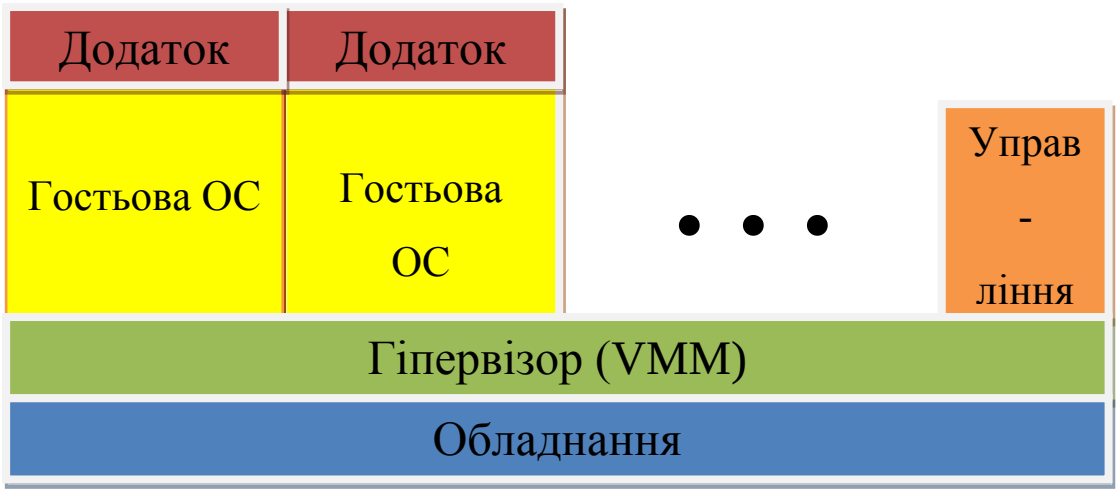


Рисунок 13.2. – Повна віртуалізація

Повна віртуалізація (Full, Native Virtualization). Використовуються не модифіковані екземпляри гостьових операційних систем, а для підтримки роботи цих ОС служить загальний шар емуляції їх виконання поверх хостової ОС, в ролі якої виступає звичайна операційна система. Така технологія застосовується, зокрема, в VMware Workstation, VMware Server (колишній GSX Server, Parallels Desktop, Parallels Server, MS Virtual PC, MS Virtual Server, Virtual Iron. До переваг даного підходу можна зарахувати відносну простоту реалізації, універсальність і надійність рішення ; всі функції управління бере на себе хост -ОС. Недоліки – високі додаткові накладні витрати на використовувані апаратні ресурси, відсутність обліку особливостей гостьових ОС, менша, ніж потрібно, гнучкість у використанні апаратних засобів.

Паравіртуалізації (paravirtualization). Модифікація ядра гостьової ОС виконується таким чином, що в неї включається новий набір API, через який

вона може безпосередньо працювати з апаратурою, не конфліктуючи з іншими віртуальними машинами. При цьому немає необхідності задіяти повноцінну ОС в якості хостового ПО, функції якого в даному випадку виконує спеціальна система, що отримала назву гіпервізора (hypervisor). Саме цей варіант є сьогодні найбільш актуальним напрямком розвитку серверних технологій віртуалізації і застосовується в VMware ESX Server, Xen (і рішеннях інших постачальників на базі цієї технології), Microsoft Hyper-V. Переваги даної технології полягають у відсутності потреби в хостової ОС – ВМ, встановлюються фактично на " голе залізо ", а апаратні ресурси

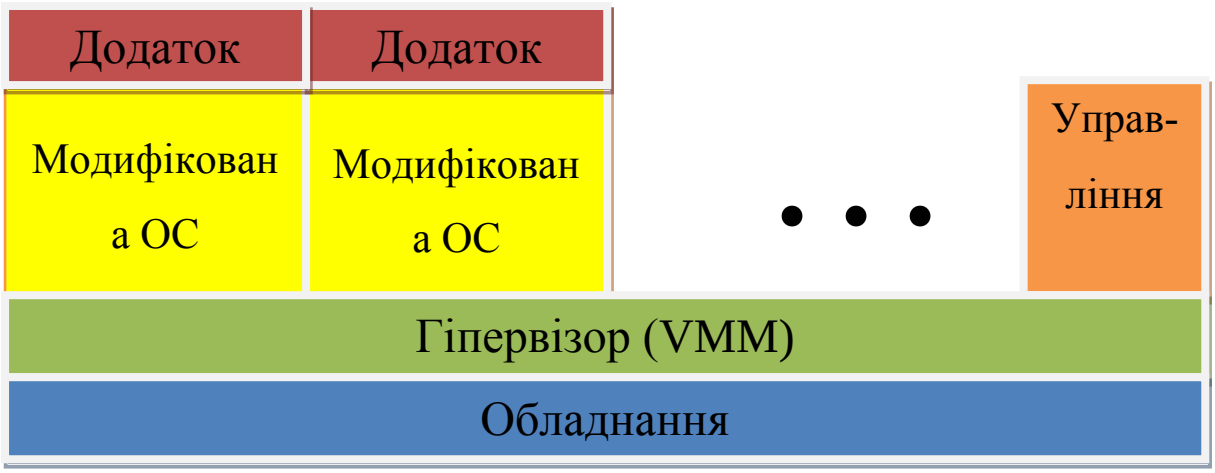


Рисунок 13.3 – Паравіртуалізації

Віртуалізація на рівні ядра ОС (operating system – level virtualization). Цей варіант передбачає використання одного ядра хостової ОС для створення незалежних паралельно працюючих операційних середовищ. Для гостьового ПО створюється тільки власне мережеве та апаратне оточення.

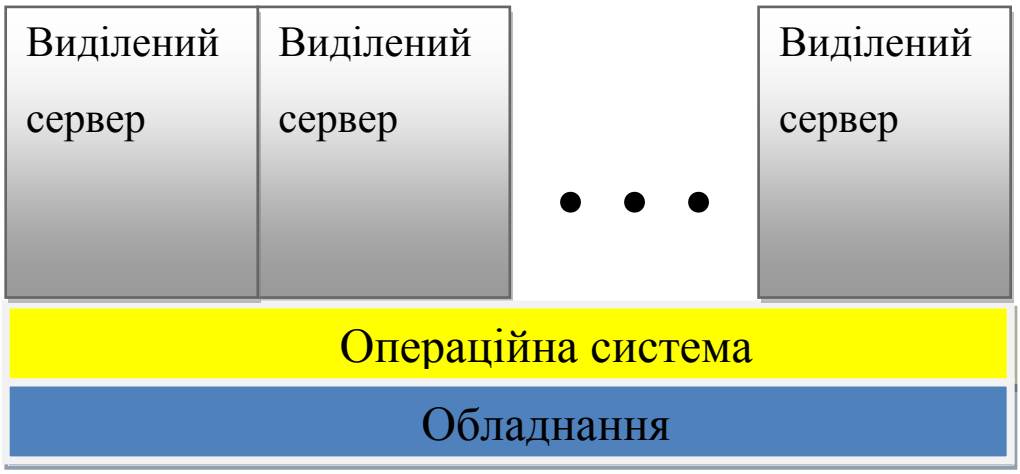


Рисунок 13.4. – Віртуалізація на рівні ОС

Такий варіант використовується в Virtuozzo (для Linux і Windows), OpenVZ (безкоштовний варіант Virtuozzo) і Solaris Containers. Переваги – висока ефективність використання апаратних ресурсів, низькі накладні технічні витрати, відмінна керованість, мінімізація витрат на придбання ліцензій. Недоліки – реалізація тільки однорідних обчислювальних середовищ.

ЛЕКЦІЯ 14. ВІРТУАЛІЗАЦІЯ ЦЕНТРІВ ОБРОБКИ ДАНИХ

14.1. Короткий огляд платформ віртуалізації

14.1.1. VMware

Компанія VMware – один з перших гравців на ринку платформ віртуалізації. У 1998 році VMware запатентувала свої програмні техніки віртуалізації і з тих пір випустила чимало ефективних і професійних продуктів для віртуалізації різного рівня: від VMware Workstation, призначеного для настільних ПК, до VMware ESX Server, що дозволяє консолідувати фізичні сервери підприємства у віртуальній інфраструктурі.

На відміну від ЕОМ (мейнфрейм) пристрої на базі x86 не підтримують віртуалізацію в повній мірі. Тому компанії VMware довелося подолати чимало проблем у процесі створення віртуальних машин для комп'ютерів на базі x86. Основні функції більшості ЦП (в ЕОМ і ПК) полягають у виконанні послідовності збережених інструкцій (тобто програм). У процесорах на базі x86 містяться 17 особливих інструкцій, що створюють проблеми при віртуалізації, через які операційна система відображає попереджувальне повідомлення, перериває роботу програми або просто видає загальний збій. Отже, ці 17 інструкцій виявилися значною перешкодою на початковому етапі впровадження віртуалізації для комп'ютерів на базі x86.

Для подолання цієї перешкоди компанія VMware розробила адаптивну технологію віртуалізації, яка " перехоплює " дані інструкції на етапі створення і перетворює їх у безпечні інструкції, придатні для віртуалізації, не зачіпаючи при цьому процеси виконання всіх інших інструкцій. У результаті ми отримуємо високопродуктивну віртуальну машину, відповідну апаратного забезпечення вузла та підтримуючу повну програмну сумісність. Компанія VMware першою розробила і впровадила дану інноваційну технологію, тому на сьогоднішній день вона є незаперечним лідером технологій віртуалізації.

У вельми великому списку продуктів VMware можна знайти чимало інструментів для підвищення ефективності та оптимізації IT – інфраструктури, управління віртуальними серверами, а також кошти міграції з фізичних платформ на віртуальні. За результатами різних тестів продуктивності засоби віртуалізації VMware майже завжди за більшістю параметрів виграють у конкурентів. VMware має більше 100 000 клієнтів по всьому світу, в списку її клієнтів 100 % організацій зі списку Fortune 100. Мережа партнерств охоплює більше 350 виробників обладнання та ПЗ і більше 6000 реселерів. На даний момент обсяг ринку, що належить VMware, оцінюється на 80 %. Тим часом, серед платформ віртуалізації у VMware є з чого вибирати.

VMware Workstation – платформа, орієнтована на desktop -користувачів і призначена для використання розробниками ПЗ, а також професіоналами у сфері IT. Нова версія популярного продукту VMware Workstation 7 стала доступна в 2009 р, в якості хостових операційних систем підтримуються ОС Windows і Linux. VMware Workstation 7 може використовуватися спільно з середовищем розробки, що робить її особливо популярною в середовищі

розробників, викладачів і фахівців технічної підтримки. Вихід VMware Workstation 7 означає офіційну підтримку Windows 7 як в якості гостьової, так і хостової операційної системи. Продукт включає підтримку Aero Peek і Flip 3D, що робить можливим спостерігати за роботою віртуальної машини, підбиваючи курсор до панелі завдань VMware або до відповідної вкладці на робочому столі хоста. Нова версія може працювати на будь -якій версії Windows 7, також як і будь -які версії Windows, можуть бути запущені у віртуальних машинах. Крім того, віртуальні машини в VMware Workstation 7, повністю підтримують Windows Display Driver Model (WDDM), що дозволяє використовувати інтерфейс Windows Aero у гостьових машинах.

VMware Player – безкоштовний «програвач» віртуальних машин на основі віртуальної машини VMware Workstation, призначений для запуску вже готових образів віртуальних машин, створених в інших продуктах VMware, а також у Microsoft VirtualPC і Symantec LiveState Recovery. Починаючи з версії 3.0 VMware Player дозволяє також створювати образи віртуальних машин. Обмеження функціональності тепер стосується в основному функцій, призначених для IT – спеціалістів та розробників ПЗ.

VMware Fusion – настільний продукт для віртуалізації на платформі Mac від компанії Apple.

VMware Server, Безкоштовний продукт VMware Server є досить потужною платформою віртуалізації, яка може бути запущена на серверах під управлінням хостових операційних систем Windows, і Linux. Основне призначення VMware Server – підтримка малих і середніх віртуальних інфраструктур невеликих підприємств. У зв'язку з невеликою складністю його освоєння і установки, VMware Server може бути розгорнутий в найкоротші терміни, як на серверах організацій, так і на комп'ютерах домашніх користувачів.

VMware ACE – продукт для створення захищених політиками безпеки віртуальних машин, які потім можна поширювати по моделі SaaS (програмне забезпечення як послуга).

VMware VSPHERE – комплекс продуктів, що представляє надійну платформу для віртуалізації ЦОД. Компанія позиціонує даний комплекс також як потужну платформу віртуалізації для створення і розгортання приватної «хмари». VMware VSPHERE поставляється в декількох випусках з можливостями, призначеними спеціально для малих компаній і середніх компаній і корпорацій.

VMware VSPHERE включає ряд компонентів, що перетворюють стандартне обладнання в загальне стійке середовище,що нагадує мейнфрейм і включає вбудовані елементи управління рівнями обслуговування для всіх додатків:

– Служби інфраструктури – це компоненти, що забезпечують всебічну віртуалізацію ресурсів серверів, сховищ і мереж, їх об'єднання та точне виділення додаткам на вимогу і відповідно до пріоритетів бізнесу.

– Служби додатків – це компоненти, що мають вбудовані елементи управління рівнями обслуговування для всіх додатків на платформі платформи VSPHERE незалежно від їх типу або ОС.

VMware Vcenter Сервер надає центральну консоль для управління віртуалізацією, що забезпечує адміністрування служб інфраструктури і додатків. Ця консоль підтримує всебічну візуалізацію всіх аспектів віртуальної інфраструктури, автоматизацію повсякденної експлуатації і масштабованість для управління великими середовищами ЦОД.

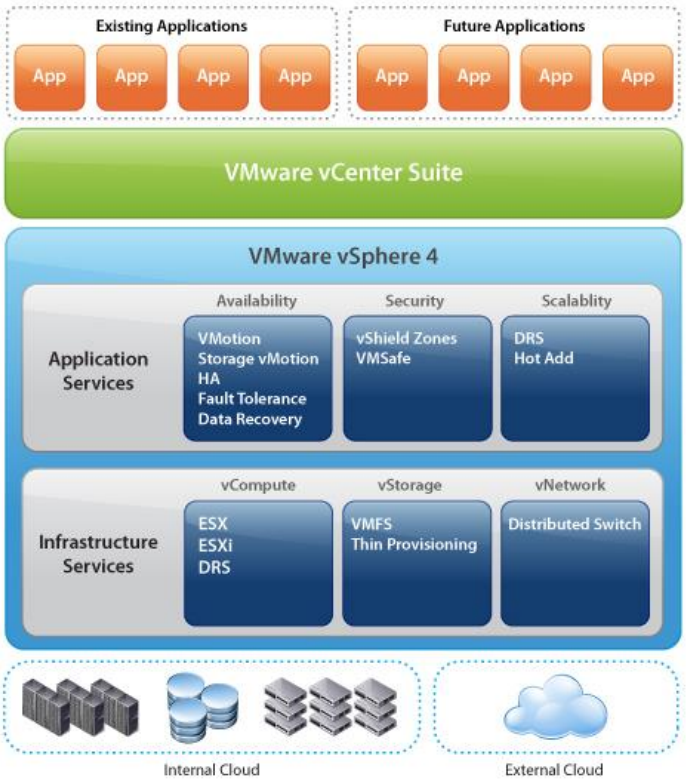


Рисунок 14.1. – Структура платформи vShpere

VMware ESX Server – це гіпервізор який розбиває фізичні сервери на безліч віртуальних машин. VMware ESX є основою пакету VMware VSPHERE і входить у всі випуски VMware VSPHERE.

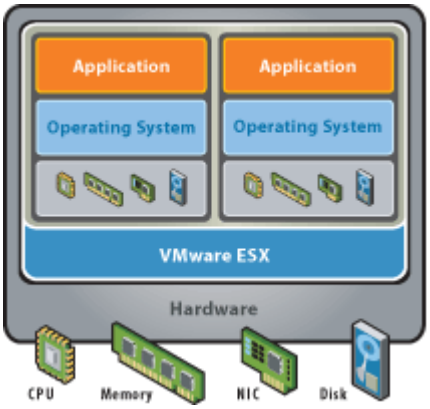


Рисунок 14.2. – Гіпервизор VMware ESX

VMware VSPHERE Hypervisor (раніше VMware ESXi) – "полегшена" платформа віртуалізації корпоративного рівня, заснована на технологіях ESX. Продукт є безкоштовним і доступний для завантаження з сайту VMware. VSphere гіпервізора VMware є найпростішим способом для початку роботи з віртуалізацією.

VMware Vcenter – надає розширювану і масштабовану платформу для попереджуючого управління віртуальною інфраструктурою і забезпечує отримання про неї всеосяжної інформації. VMware Vcenter сервер забезпечує централізоване управління середовищами VSPHERE і спрощує виконання повсякденних завдань, значно покращуючи адміністративне управління середовищем. Продукт має широкі можливості по консолідації серверів, їх налаштування та управління. VMware Vcenter сервер агрегує в собі всі аспекти управління віртуальним середовищем: від віртуальних машин до збору інформації про фізичних серверах для подальшої їх міграції у віртуальну інфраструктуру. Крім центрального продукту управління віртуальною інфраструктурою Vcenter сервера існує також ряд доповнень, що реалізують різні аспекти планування, управління та інтеграції розподіленої віртуальної інфраструктури (серверів VMware Vcenter серцебиття, VMware Vcenter Orchestrator, VMware Vcenter Ємність IQ, VMware Site Recovery Vcenter менеджер VMware Lab Manager Vcenter, VMware Vcenter Configuration Manager, VMware Vcenter перетворювач). Зокрема, Vcenter Конвертор призначений для перекладу у віртуальне середовище фізичних серверів, що дозволяє здійснювати «гарячу» (без зупину систем) та «холодну» міграцію. Vcenter Site Recovery Manager – це ПЗ для створення територіально – віддаленого резервного сегмента віртуальної інфраструктури, який у разі відмови основного вузла, бере на себе функції по запуску віртуальних машин у відповідності з планом відновлення після збоїв. Vcenter Lab Manager – продукт для створення інфраструктури зберігання і доставки конфігурацій віртуальних машин, що дозволяє організувати ефективну схему тестування в компаніях – розробниках ПЗ.

VMware ThinApp – колишній продукт Thinstall Virtualization Люкс, ПЗ для віртуалізації додатків, що дозволяє поширювати встановлені додатки на клієнтські робочі станції, скорочуючи час на стандартні операції з встановлення та конфігурації.

VMware View – комплекс продуктів, що забезпечує централізацію користувача робочих станцій у віртуальних машинах на платформі VSPHERE. Це дозволяє скоротити витрати на стандартні IT – операції, пов'язані з розгортанням та обслуговуванням користувальницьких десктопів.

VMware Capacity Planner – засіб централізованого збору та аналізу даних про апаратне і програмне забезпечення серверів, а також продуктивності обладнання. Ці дані використовуються авторизованими партнерами VMware для побудови планів консолідації віртуальних машин на платформі VMware ESX Server.

VMware VMmark – продукт, доступний тільки виробникам апаратного забезпечення, призначений для тестування продуктивності VMware ESX Server на серверних платформах.

14.1.2. Citrix (Xen)

Розробка некомерційного гіпервізора Xen починалася як дослідницький проект комп'ютерної лабораторії Кембриджського університету. Засновником проекту та його лідером був Іан Пратт (Ian Pratt) співробітник університету, який створив згодом компанію XenSource, що займається розробкою комерційних платформ віртуалізації на основі гіпервізора Xen, а також підтримкою Open Source співтовариства некомерційного продукту Xen. Спочатку Xen представляв собою найрозвиненішу платформу, що підтримує технологію паравіртуалізації. Ця технологія дозволяє гіпервізорами в хостової системі керувати гостьовий ОС допомогою гіпервизовов ДМС (Virtual – машинний інтерфейс), що вимагає модифікації ядра гостьової системи. На даний момент безкоштовна версія Xen включена в дистрибутиви декількох ОС, таких як Red Hat, Novell SUSE, Debian, Fedora Core, Sun Solaris. У середині серпня 2007 року компанія XenSource була поглинена компанією Citrix Systems. Сума проведеної операції близько 500 мільйонів доларів (акціями та грошовими коштами) говорить про серйозні наміри Citrix щодо віртуалізації. Експерти вважають, що не виключена і покупка Citrix компанією Microsoft, враховуючи її давню співпрацю з XenSource.

Безкоштовний Xen. В даний час Open Source версія платформи Xen застосовується в основному в освітніх і дослідницьких цілях. Деякі вдалі ідеї, реалізовані численними розробниками з усього світу, знаходять своє відображення в комерційних версіях продуктів віртуалізації компанії Citrix. Зараз безкоштовні версії Xen включаються до дистрибутиви багатьох Linux – систем, що дозволяє їх користувачам застосовувати віртуальні машини для ізоляції програмного забезпечення в гостьових ОС з метою його тестування і вивчення проблем безпеки, без необхідності установки платформи віртуалізації. До того ж багато незалежні розробники ПЗ можуть поширювати його за допомогою віртуальних шаблонів, в яких вже встановлена і налаштована гостьова система і пропонований продукт. Крім того, Xen ідеально підходить для підтримки старого програмного забезпечення у віртуальній машині. Для більш серйозних цілей у виробничому середовищі підприємства необхідно використовувати комерційні платформи компанії Citrix.

Citrix XenApp – продукт, призначений для віртуалізації та публікації додатків в цілях оптимізації інфраструктури доставки сервісів у великих компаніях. XenApp має величезну кількість користувачів по всьому світу і в багатьох компаніях є ключовим компонентом IT -інфраструктури.

Citrix XenServer – платформа для консолідації серверів підприємств середнього масштабу, що включає основні можливості для підтримки віртуальної інфраструктури. Виробник позиціонує даний продукт як рішення корпоративного рівня для віртуалізації серверів, що підтримує роботу в «хмарному» оточенні.

Citrix XenDesktop – рішення з віртуалізації десктопів підприємства, що дозволяє централізовано зберігати і доставляти робочі оточення у віртуальних машинах користувачам. Продукт підтримує кілька сценаріїв доставки додатків

на настільні ПК, тонкі клієнти і мобільні ПК і сумісний з серверними віртуалізаційними рішеннями конкурентів.

14.1.3. Microsoft

Для Microsoft все почалося, коли в 2003 році вона придбала компанію Connectix, одну з небагатьох компаній виробляє програмне забезпечення для віртуалізації під Windows. Разом з Connectix, компанії Microsoft дістався продукт Virtual PC,що конкурував тоді з розробками компанії VMware щодо настільних систем віртуалізації. За великим рахунком, Virtual PC надавав тоді таку кількість функцій, що і VMware Workstation, і при належній увазі міг би бути в даний час повноцінним конкурентом цієї платформи. Проте з того часу, компанія Microsoft випускала по мінорному релізу на рік, не приділяючи особливої уваги продукту Virtual PC, в той час як VMware стрімко розвивала свою систему віртуалізації, перетворивши її по – справжньому в професійний інструмент. Усвідомивши своє технологічне відставання у сфері віртуалізації серверних платформ, компанія Microsoft випустила продукт Virtual Server 2005, націлений на створення і консолідацію віртуальних серверів організацій. Однак було вже пізно. Компанія VMware вже захопила лідерство в цьому сегменті ринку, пропонуючи в той момент дві серверні платформи віртуалізації VMware GSX сервера і VMware ESX Server, кожна з яких за багатьма параметрами перевершувала платформу Microsoft. Остаточний удар був нанесений в 2006 році, коли VMware фактично оголосила продукт VMware GSX сервера безкоштовним, взявшись за розробку продукту VMware Server на його основі і сконцентрувавши всі зусилля на продажах потужної корпоративної платформи VMware ESX Server у складі віртуальної інфраструктури Virtual Infrastructure 3.

У компанії Microsoft був тільки єдиний вихід у цій ситуації: у квітні 2006 року вона також оголосила про безкоштовність продукту Microsoft Virtual Server 2005. Також існуючі раніше два видання Standard Edition і Enterprise Edition були об'єднані в одне – Microsoft Virtual Server Enterprise Edition. З тих пір Microsoft істотно змінила стратегію щодо віртуалізації, і влітку 2008 року був випущений фінальний реліз платформи віртуалізації Microsoft Hyper -V, інтегрованої в ОС Windows Server 2008. Тепер роль сервера віртуалізації доступна всім користувачам нової серверної операційної системи Microsoft.

Microsoft Virtual Server. Серверна платформа віртуалізації Microsoft Virtual Server може використовуватися на сервері під управлінням операційної системи Windows Server 2003 і призначена для одночасного запуску декількох віртуальних машин на одному фізичному хості. Платформа безкоштовна і надає тільки базові функції.

Microsoft Virtual PC. Продукт Virtual PC був куплений корпорацією Microsoft разом з компанією Connectix і вперше під маркою Microsoft був випущений як Microsoft Virtual PC 2004. Купуючи Virtual PC і компанію Connectix, компанія Microsoft будувала далекосяжні плани щодо забезпечення користувачів інструментом для полегшення міграції на наступну версію операційної системи Windows. Тепер Virtual PC 2007 безкоштовний і доступний для підтримки настільних ОС у віртуальних машинах.

Microsoft Hyper -V. Продукт Microsoft позиціонується як основний конкурент VMware ESX Server в області корпоративних платформ віртуалізації. Microsoft Hyper -V являє собою рішення для віртуалізації серверів на базі процесорів з архітектурою x64 в корпоративних середовищах. На відміну від продуктів Microsoft Virtual Server або Virtual PC, Hyper -V забезпечує віртуалізацію на апаратному рівні, з використанням технологій віртуалізації, вбудованих в процесори. Hyper -V забезпечує високу продуктивність, практично рівну продуктивності однієї операційної системи, що працює на виділеному сервері. Hyper -V поширюється двома способами: як частина Windows Server 2008 або у складі незалежного безкоштовного продукту Microsoft Hyper – V Server.

У Windows Server 2008 технологія Hyper -V може бути розгорнута як у повній установці, так і в режимі Server Core, Hyper – V Server працює тільки в режимі Core. Це дозволяє повною мірою реалізувати всі переваги «тонкої», економічною і керованої платформи віртуалізації.

Hyper -V є вбудованим компонентом 64 – розрядних версій Windows Server 2008 Standard, Windows Server 2008 Enterprise і Windows Server 2008 Datacenter. Ця технологія недоступна в 32 – розрядних версіях Windows Server 2008, в Windows Server 2008 Standard без Hyper -V, Windows Server 2008 Enterprise без Hyper -V, Windows Server 2008 Datacenter без Hyper -V, в Windows Web Server 2008 і Windows Server 2008 для систем на базі Itanium.

Розглянемо коротко особливості архітектури Hyper – v. Hyper – v являє собою гіпервізор, тобто прошарок між обладнанням та віртуальними машинами рівнем нижче операційної системи. Ця архітектура була спочатку розроблена IBM в 1960 -і роки для мейнфреймів і нещодавно стала доступною на платформах x86/x64, як частина низки рішень, включаючи Windows Server 2008 Hyper – V і Vmware ESX.

Рисунок 14.3. – Архітектура віртуалізації з гіпервізором

Віртуалізація на базі гіпервізора заснована на тому, що між обладнанням та віртуальними машинами з'являється прошарок, що перехоплює звернення операційних систем до процесора, пам'яті і інших пристроїв. При цьому доступ до периферійних пристроїв у різних реалізаціях гіпервізорів може бути організований по – різному. З точки зору існуючих рішень для реалізації менеджера віртуальних машин можна виділити два основних види архітектури гіпервізора: мікроядерну і монолітну.

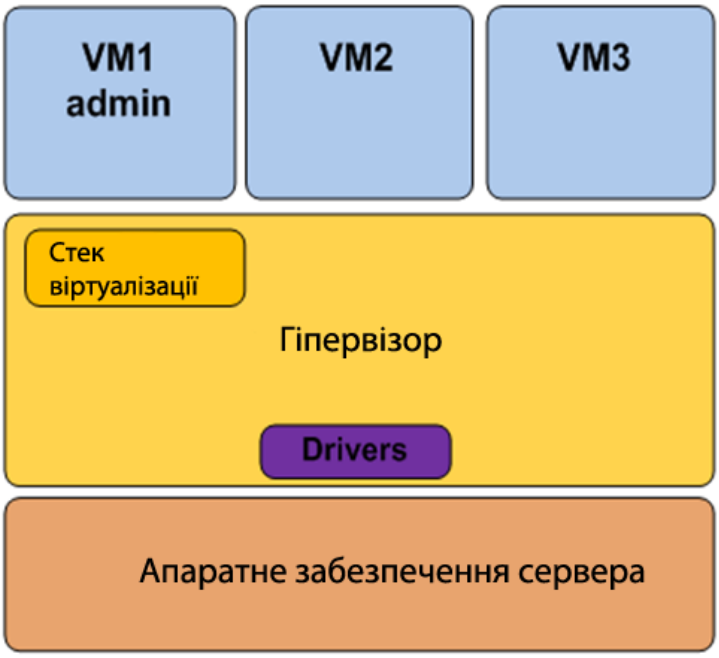


Рисунок 13.4. – Архітектура монолітного гіпервізора

Монолітний підхід розміщує гіпервізор в єдиному рівні, який також включає більшість необхідних компонентів, таких як ядро, драйвери пристроїв і стек введення / виведення. Це підхід, який використовується такими рішеннями, як VMware ESX і традиційні системи мейнфреймів.

Монолітний підхід має на увазі, що всі драйвери пристроїв поміщені в гіпервізор. У монолітної моделі – для доступу до обладнання гіпервізор використовуються власні драйвери. Гостьові ОС працюють на віртуальних машинах поверх гіпервізора. Коли гостьовий системі потрібен доступ до устаткування, вона повинна пройти через гіпервізор і його модель драйверів. Зазвичай одна з гостьових ОС грає роль адміністратора або консолі, у якому запускаються компоненти для надання ресурсів, управління і моніторингу всіх гостьових ОС, що працюють на сервері.

Модель монолітного гіпервізора забезпечує прекрасну продуктивність, але має ряд недоліків, таких як:

– Стійкість – якщо в оновленій версії драйвера затесалася помилка, в результаті збої почнуться у всій системі, у всіх її віртуальних машинах.

– Проблеми оновлення драйверів – при необхідності оновлення драйвера якого -небудь пристрою (наприклад мережного адаптера) оновити драйвер можливо тільки разом з виходом нової версії гіпервізора, в яку буде інтегрований новий драйвер для цього пристрою.

– Труднощі з використанням непідтримуваного обладнання. Наприклад, ви зібралися використовувати обладнання «Сервер» досить потужний і надійний, але при цьому в гіпервізора не виявилося потрібного драйвера для RAID – контролера або мережного адаптера. Це зробить неможливим використання відповідного обладнання, а, значить, і сервера.

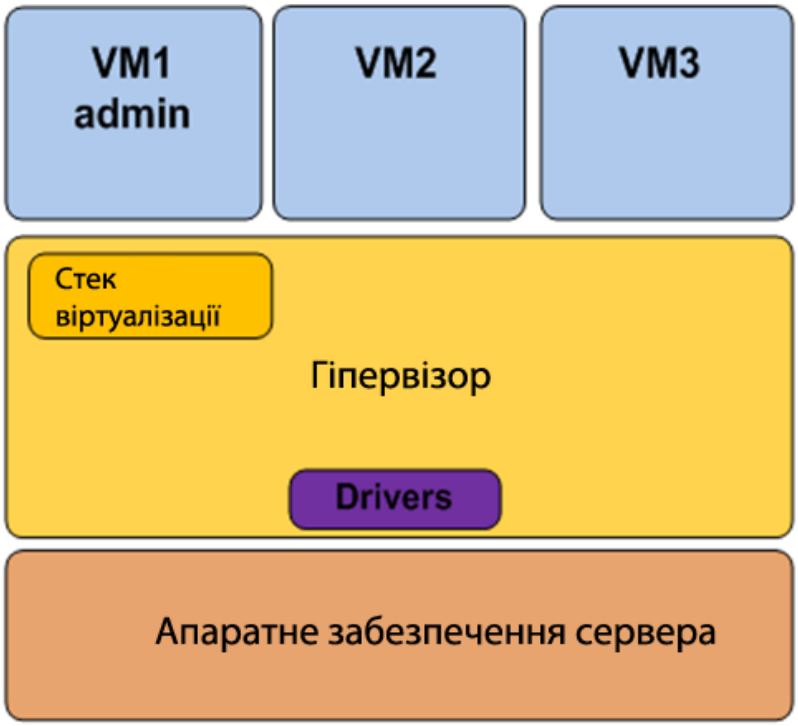


Рисунок 14.5. – Архітектура мікроядерного гіпервізора

Мікроядерний підхід використовує дуже тонкий, спеціалізований гіпервізор, що виконує лише основні задачі забезпечення ізоляції розділів і управління пам'яттю. Цей рівень не включає стеку введення / виводу або драйверів пристроїв. Це підхід, який використовується у Hyper -V. У цій архітектурі стек віртуалізації і драйверів конкретних пристроїв розташовані в спеціальному розділі ОС, іменованому батьківським розділом.

У мікроядерній реалізації можна говорити про «тонкий гіпервізор», в цьому випадку в ньому зовсім немає драйверів. Замість цього драйвери працюють в кожному індивідуальному розділі, щоб будь -яка гостьова ОС мала можливість отримати через гіпервізор доступ до обладнання. При такій розстановці сил кожна віртуальна машина займає цілком відокремлений розділ, що позитивно позначається на захищеності і надійності. У мікроядерної моделі гіпервізора (у віртуалізації Windows Server 2008 R2 використовується саме вона) один розділ є батьківським (parent), решта – дочірніми (child). Розділ – це найменша ізольована одиниця, підтримувана гіпервізором. Розмір гіпервізора Hyper -V менше 1,5 Мб, він може поміститися на одну 3.5 – дюймову дискету.

Кожному розділу призначаються конкретні апаратні ресурси – частку процесорного часу, обсяг пам'яті, пристрої та пр. Батьківський розділ створює дочірні розділи і керує ними, а також містить стек віртуалізації (virtualization stack), використовуваний для управління дочірніми розділами. Батьківський розділ створюється першим і володіє всіма ресурсами, що не належать гіпервізорами. Володіння всіма апаратними ресурсами означає, що саме корінний (тобто, батьківський) розділ управляє живленням, підключенням самоналагоджувальних пристроїв, відає питаннями апаратних збоїв і навіть управляє завантаженням гіпервізора.

У батьківському розділі міститься стек віртуалізації – набір програмних компонентів, розташованих поверх гіпервізора котрі разом з ним забезпечують роботу віртуальних машин. Стек віртуалізації обмінюється даними з гіпервізором і виконує всі функції з віртуалізації, які не підтримуються безпосередньо гіпервізором. Велика частина цих функцій пов'язана із створенням дочірніх розділів та управлінням ними і необхідними їм ресурсами (ЦП, пам'ять, пристрої).

Перевага мікроядерного підходу, застосованого в Windows Server 2008 R2, порівняно з монолітним підходом полягає в тому, що драйвери, які повинні розташовуватися між батьківським розділом і фізичним сервером, не потребують внесення жодних змін в модель драйверів. Іншими словами, у системі можна просто застосовувати існуючі драйвери. У Microsoft обрали цей підхід, оскільки необхідність розробки нових драйверів сильно загальмувала б розвиток системи. Що ж стосується гостьових ОС, вони будуть працювати з емуляторами або синтетичними пристроями.

З іншого боку, що мікроядерна модель може дещо програвати монолітної моделі в продуктивності. Однак у наші дні головним пріоритетом стала безпека, тому для більшості компаній цілком прийнятна буде втрата пари відсотків в продуктивності заради скорочення фронту нападу і підвищення стійкості.

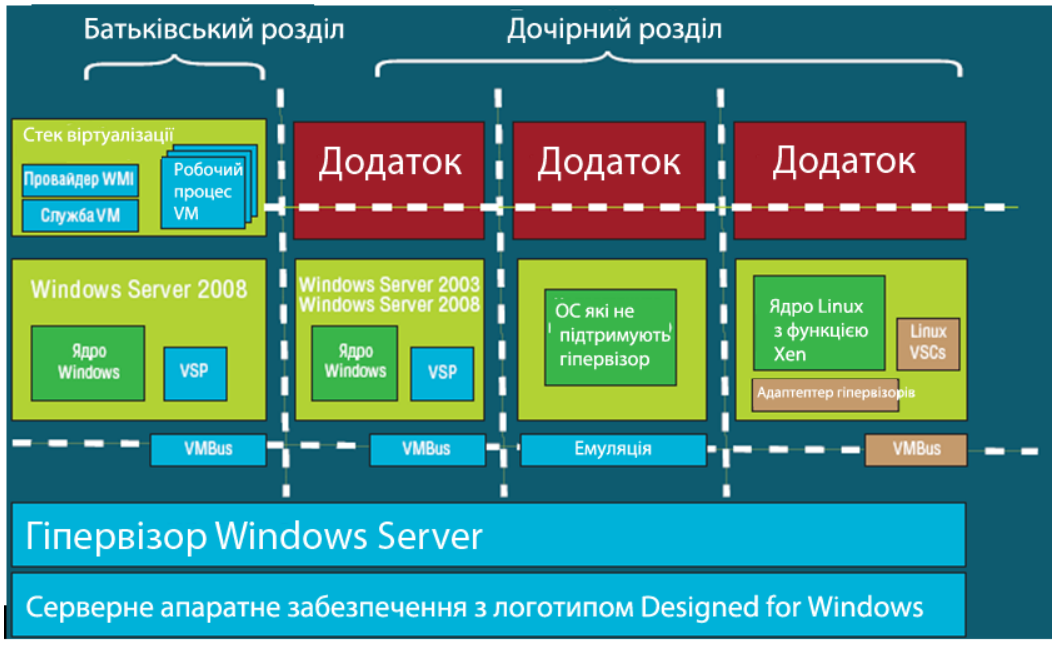


Рисунок 14.6. – Архітектура Hyper – v

Всі версії Hyper – V мають один батьківський розділ. Цей розділ керує функціями Hyper -V. З батьківського розділу запускається консоль Windows Server Virtualization. Крім того, батьківський розділ використовується для запуску віртуальних машин (VM), що підтримують потокову емуляцію старих апаратних засобів. Такі VM, побудовані на готових шаблонах, емулює апаратні засоби, є аналогами VM, що працюють в продуктах з віртуалізацією на базі хосту, наприклад Virtual Server.

Гостьові VM запускаються з дочірніх розділів Hyper -V. Дочірні розділи підтримують два типи VM: високопродуктивні VM на основі архітектури VMBus і VM, керовані системою – хостом. У першу групу входять VM з системами Windows Server 2003, Windows Vista, Server 2008 і Linux (підтримуючими Xen). Нову архітектуру VMBus відрізняє високопродуктивний конвеєр, функціонуючий в оперативній пам'яті, що з'єднує клієнтів Virtualization Service Clients (VSC) на гостьових VM з провайдером Virtual Service Provider (VSP) хоста. VM, керовані хостом, запускають платформи, які не підтримують нову архітектуру VMBus: Windows NT, Windows 2000 і Linux (без підтримки технології Xen, наприклад SUSE Linux Server Enterprise 10).

Microsoft System Center Virtual Machine Manager (SCVMM) – окремий продукт сімейства System Center для управління віртуальною інфраструктурою, ефективного використанням ресурсів фізичних вузлів, а також спрощення підготовки та створення нових гостьових систем для адміністраторів і користувачів. Продукт забезпечує всебічну підтримку консолідації фізичних серверів у віртуальній інфраструктурі, швидке та надійне перетворення фізичних машин у віртуальні, розумне розміщення віртуальних навантажень на відповідних фізичних вузлах, а також єдину консоль для управління ресурсами та їх оптимізації. SCVMM забезпечує наступні можливості:

– Централізоване управління серверами віртуальних машин в масштабах підприємства. SCVMM підтримує управління серверами Microsoft Hyper -V, Microsoft Virtual Server, VMware ESX і в майбутньому буде реалізована підтримка Xen.

– Створення бібліотеки шаблонів віртуальних машин. Шаблони віртуальних машин представляють собою набори образів встановлених операційних систем, які можуть бути розгорнуті за лічені хвилини.

– Моніторинг та розміщення віртуальних машин у відповідність із завантаженістю фізичних серверів.

– Міграція (конвертація) фізичних серверів у віртуальні машини – технологія P2V. Технологія P2V дозволяє зробити перенесення фізичного сервера на віртуальний без зупинки роботи. Таким чином, з'являється можливість онлайнового резервування цілого сервера, і в разі виходу його з ладу, можна протягом хвилини запустити віртуальний сервер і продовжити роботу.

– Міграція (конвертація) віртуальних машин інших форматів у віртуальні машини Hyper -V – технологія V2V. Дана технологія аналогічна P2V, але при цьому дозволяє переносити віртуальні машини Microsoft Virtual Server або VMware ESX в Hyper -V.

– Управління кластерами Hyper -V.

Ключові терміни:

Віртуалізація – процес подання набору обчислювальних ресурсів або їх логічного об'єднання, який дає певні переваги перед оригінальною конфігурацією.

Віртуальна машина – програмне або апаратне середовище, яка приховує справжню реалізацію якогось процесу чи об'єкту залежно від його видимого подання.

Повна віртуалізація – віртуалізація при якій використовуються не модифіковані екземпляри гостьових операційних систем, а для підтримки роботи цих ОС служить загальний шар емуляції їх виконання поверх хостової ОС, в ролі якої виступає звичайна операційна система.

Паравіртуалізаціїя – віртуалізація при якій проводиться модифікація ядра гостьової ОС виконується таким чином, що в неї включається новий набір API, через який вона може безпосередньо працювати з апаратурою, що не конфліктуючи з іншими віртуальними машинами.

Віртуалізація на рівні ОС – вид віртуалізації, який передбачає використання одного ядра хостової ОС для створення незалежних паралельно працюючих операційних середовищ.

Віртуалізація серверів – це запуск на одному фізичному сервері декількох віртуальних серверів. Віртуальні машини або сервера являють собою програми, запущені на хостовій операційній системі, які емулюють фізичні пристрої сервера. На кожній віртуальній машині може бути встановлена операційна система, на яку можуть бути встановлені додатки і служби.

Віртуалізація додатків – вид віртуалізації, який передбачає застосування моделі сильної ізоляції прикладних програм з керованим взаємодією з ОС, при якому віртуалізується кожен екземпляр додатків, всі його основні компоненти: файли (включаючи системні), реєстр, шрифти, INI – файли, COM -об'єкти, служби. Додаток виповнюється без процедури інсталяції в традиційному її розумінні і може запускатися прямо з зовнішніх носіїв.

Віртуалізація уявлень (робочих місць) Віртуалізація уявлень має місце, коли сервер надає свої ресурси клієнтам, причому клієнтську програму виконується на цьому сервері, а клієнт отримує лише подання.

Монолітна архітектура гіпервізора – архітектура гіпервізора при якій гіпервізор розміщується на єдиному рівні, який також включає більшість необхідних компонентів, таких як ядро, драйвери пристроїв і стек введення / виведення

Мікроядерна архітектура гіпервізора – підхід при якому використовується дуже тонкий, спеціалізований гіпервізор, що виконує лише основні задачі забезпечення ізоляції розділів і управління пам'яттю. Цей рівень не включає стека введення / виводу або драйверів пристроїв.

Литература:

1. http://www.pcweek.ru/themes/detail.php?ID=107230
2. http://www.ixbt.com/cm/virtualization.shtml
3. Александр Самойленко «Обзор популярных платформ виртуализации VMware, Citrix и Microsoft»
4. http://www.vmgu.ru

ЛЕКЦІЯ 15. GRID І БАЗИ ДАНИХ. УПРАВЛІННЯ GRID-ОТОЧЕННЯМ

15.1. Розподілені БД

Основною причиною розробки систем, що використовують бази даних, являється прагнення інтегрувати усі оброблювані в організації дані в єдине ціле і забезпечити до них контрольований доступ. Хоча інтеграція і надання контрольованого доступу можуть сприяти централізації, остання не являється самоціллю. На практиці створення комп'ютерних мереж призводить до децентралізації обробки даних. Децентралізований підхід, по суті, відбиває організаційну структуру компанії, що логічно складається з окремих підрозділів, відділів, проектних груп і тому подібного, які фізично розподілені по різних офісах, відділеннях, підприємствах або філіях, причому кожна окрема одиниця має справу з власним набором оброблюваних даних. Розробка розподілених баз даних, що відбивають організаційні структури підприємств, дозволяє зробити дані, підтримувані кожним з існуючих підрозділів, загальнодоступними, забезпечивши при цьому їх збереження саме в тих місцях, де вони найчастіше використовуються. Подібний підхід розширює можливості спільного використання інформації, одночасно підвищуючи ефективність доступу до неї.

Розподілені системи покликані вирішити проблему островів інформації. Бази даних іноді розглядають як деякі електронні острови. Це положення може бути наслідком географічної роз'єднаності, несумісності використовуваної комп'ютерної архітектури, несумісності використовуваних комутаційних протоколів і так далі. Інтеграція окремих баз даних в одне логічне ціле здатна змінити подібний стан справ.

Щоб почати обговорення пов'язаних з розподіленими СКБД проблем, передусім необхідно утямити, що ж таке розподілена база даних.

Розподілена база даних – Набір логічно пов'язаних між собою даних (і їх описів), що розділяються, які фізично розподілені в деякій комп'ютерній мережі.

З цього витікає наступне визначення.

Розподілена СКБД – Програмний комплекс, призначений для управління розподіленими базами даних і який дозволяє зробити розподіленість інформації прозорою для кінцевого користувача.

Система управління розподіленими базами даних (СКРБД) складається з єдиної логічної бази даних, розділеної на деяку кількість фрагментів. Кожен фрагмент бази даних зберігається на одному або декількох комп'ютерах, які сполучені між собою лініями зв'язку і кожен з яких працює під керуванням окремої СКБД. Будь-який з сайтів здатний незалежно обробляти запити користувачів, що вимагають доступу до даних (що створює певну міру локальної автономії), які зберігаються локально, а також здатний обробляти дані мережі, що зберігаються на інших комп'ютерах мережі.

Користувачі взаємодіють з розподіленою базою даних через програми (застосування). Застосування можуть бути класифіковані як ті, які не вимагають доступу до даних на інших сайтах (локальні застосування), і ті, які вимагають подібного доступу (глобальні застосування). У розподіленій СКБД повинно існувати хоча б одне глобальне застосування, тому будь-яка СКРБД повинна мати наступні особливості.

– Набір логічно пов'язаних розподіоених даних.

– Дані, що зберігаються, розбиті на деяку кількість фрагментів.

– Між фрагментами може бути організована реплікація даних. Фрагменти і їх репліки розподілені по різних сайтах.

– Сайти пов'язані між собою мережевими з'єднаннями.

– Робота з даними на кожному сайті управляється СКБД.

– СКБД на кожному сайті здатна підтримувати автономну роботу локальних застосувань.

– СКБД кожного сайту підтримує хоч би одне глобальне застосування.

Немає необхідності в тому, щоб на кожному з сайтів системи існувала своя власна локальна база даних, що і показане на прикладі топології СКРБД, представленої на рисунку 15.1.

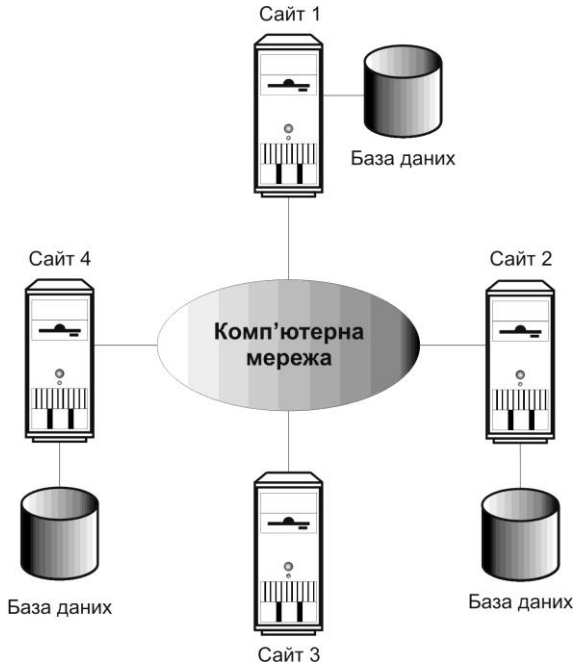


Рисунок 15.1. – Топологія системи управління розподіленою базою даних

Приклад розподіленої обробки даних.

Використовуючи технологію розподілених баз даних, компанія замість єдиного центрального комп'ютера може розмістити свою базу даних на декількох незалежних комп'ютерних системах. Подібні комп'ютерні системи можуть бути встановлені в кожному з існуючих відділень компанії у різних місцях. Мережеві з'єднання, що зв'язують комп'ютерні системи, дозволять відділенням компанії взаємодіяти між собою, а розгортання в системі СКРБД

забезпечить доступ до даних, розміщених в інших відділеннях компанії. В результаті клієнт, що проживає в одному місті, зможе звернутися в найближче відділення компанії і ознайомитися з об'єктами нерухомості, що надаються в оренду в іншому населеному пункті, що позбавить його від необхідності для отримання цих відомостей зв'язуватися з потрібним містом по телефону або посилати поштове повідомлення.

З визначення СКРБД виходить, що для кінцевого користувача розподіленість системи має бути абсолютно прозора. Іншими словами, від користувачів має бути повністю прихований той факт, що розподілена база даних складається з декількох фрагментів, які можуть розміщуватися на різних комп'ютерах і для яких, можливо, організована служба реплікації даних. Призначення забезпечення прозорості полягає в тому, щоб розподілена система зовні поводилася точно так, як і централізована. В деяких випадках цю вимогу називають **основним принципом побудови розподілених СКБД**. Цей принцип вимагає надання кінцевому користувачеві суттєвого діапазону функціональних можливостей, але, на жаль, одночасно ставить перед програмним забезпеченням СКРБД безліч додаткових завдань, з якими ми детальніше ознайомимося далі.

Розподілена обробка

Дуже важливо розуміти відмінності, існуючі між розподіленими СКБД і розподіленою обробкою даних.

Розподілена обробка – Обробка з використанням централізованої бази даних, доступ до якої може здійснюватися з різних компьютерів мережі.

Паралельні СКБД

Крім того, слід чітко розуміти відмінності, існуючі між розподіленими і паралельними СКБД.

Паралельна СКБД – Система управління базою даних, що функціонує з використанням декількох процесорів і пристроїв жорстких дисків, що дозволяє їй (якщо це можливо) розпаралелювати виконання деяких операцій з метою підвищення загальної производи-тельности обробки.

Переваги і недоліки, властиві розподіленим СКБД

Системи з розподіленими базами даних мають додаткові переваги перед традиційними централізованими системами баз даних. На жаль, ця технологія не позбавлена і деяких недоліків. У цьому розділі ми обговоримо як переваги, так і недоліки, властиві системам з розподіленими базами даних.

Переваги

Віддзеркалення структури організації

Подільність і локальна автономність

Підвищення доступності даних

У централізованих СКБД відмова центрального комп'ютера викликає припинення функціонування усієї СКБД. Проте відмова одного з сайтів СКРБД або лінії зв'язку між сайтами зробить недоступною лише деякі сайти, тоді як уся система в цілому збереже свою працездатність. Розподілені СКБД проектуються так, щоби забезпечувати продовження функціонування системи,

незважаючи на подібні відмови. Якщо виходить з ладу один з вузлів, система зможе перенаправити запити до вузла, що відмовив, на адресу іншого сайту.

Підвищення надійності

Якщо організована реплікація даних, внаслідок чого дані і їх копії будуть розміщені на більш, ніж одному сайті, відмова окремого вузла або сполучного зв'язку між вузлами не приведе до недоступності даних в системі.

Підвищення продуктивності

Якщо дані розміщені на самому навантаженому сайті, який успадкував від систем-попередників високий рівень паралельності обробки, то розгортання розподіленої СКБД може сприяти підвищенню швидкості доступу до бази даних (в порівнянні з доступом до віддаленої централізованої СКБД). Більше того, оскільки кожен сайт працює тільки з частиною бази даних, рівень використання центрального процесора і служб вводу/виводу може виявитися нижчим, ніж у разі централізованої СКБД.

Економічні вигоди

Виявляється, що набагато вигідніше встановлювати в підрозділах організації власні малопотужні комп'ютери, крім того, значно дешевше додати в мережу нові робочі станції, ніж модернізувати систему з центарльним сервером.

Друге потенційне джерело економії має місце у тому випадку, коли бази даних географічно віддалені одна від одної і застосування вимагають здійснення доступу до розподілених даних. В цьому випадку через відносно високу вартість передаваних по мережі даних (в порівнянні з вартістю їх локальною обробки) може виявитися економічно вигідним розділити застосування на відповідні частині і виконувати необхідну обробку на кожному з сайтів локально.

Модульність системи

У розподіленому середовищі розширення існуючої системи здійснюється набагато простіше. Додавання в мережу нового сайту не чинить впливу на функціонування вже існуючих. Подібна гнучкість дозволяє організації легко розширюватися. Перевантаження через збільшення розміру бази даних зазвичай усуваються шляхом додавання в мережу нових обчислювальних потужностей і пристроїв дискової пам'яті. У централізованих СКБД зростання розміру бази даних може вимагати заміни і устаткування (потужнішою системою), і використовуваного програмного забезпечення (потужнішою або гнучкішою СКБД).

Недоліки

Підвищення складності

Розподілені СКБД, здатні приховати від кінцевих користувачів розподілену природу використовуваних ними даних і забезпечити необхідний рівень продуктивності, надійності і доступності, безумовно є складнішими програмними комплексами, ніж централізовані СКБД. Той факт, що дані можуть піддаватися реплікації, також додає додатковий рівень складності в програмне забезпечення СКРБД. Якщо реплікація даних не буде підтримуватися на необхідному рівні, система матиме нижчий рівень

доступності даних, надійності і продуктивності, ніж централізовані системи, а усі викладені вище переваги перетворяться на недоліки.

Збільшення вартості

Збільшення складності означає і збільшення витрат на придбання і супровід СКРБД (в порівнянні із звичайними централізованими СКБД). Разворачива-ние розподіленої СКБД зажадає додаткового устаткування, необхідного для установки мережевих з'єднань між сайтами. Слід чекати і зростання затрат на оплату каналів зв'язку, викликаних зростанням мережевого трафіку. Крім того, зростуть витрати на оплату праці персоналу, який буде потрібно для обслуговування локальних СКБД і мережевих з'єднань.

Проблеми захисту

У централізованих системах доступ до даних легко контролюється. Проте в розподілених системах потрібно буде організувати контроль доступу не лише до даних, реплікованих на декілька різних сайтів, але і захист мережевих з'єднань самих по собі. Раніше мережі розглядалися як абсолютно незахищені канали зв'язку. Хоча це частково справедливо і для теперішнього часу, проте відносно захисту мережевих з'єднань досягнутий дуже істотний прогрес.

Ускладнення контролю за цілісністю даних

Цілісність бази даних означає коректність і узгодженість даних, що зберігаються в ній. Вимоги забезпечення цілісності зазвичай формулюються у вигляді деяких обмежень, виконання яких гарантуватиме захист інформації в базі даних від пошкодження. Реалізація обмежень підтримки цілісності зазвичай вимагає доступу до великої кількості даних, використовуваних при виконанні перевірок, але не вимагає виконання операцій оновлення. У розподілених СКБД підвищена вартість передачі і обробки даних може перешкоджати організації ефективного захисту від порушень цілісності даних.

Відсутність стандартів

Хоча цілком очевидно, що функціонування розподілених СКБД залежить від ефективності використовуваних каналів зв'язку, тільки останнім часом стали вимальовуватися контури стандарту на канали зв'язку і протоколи доступу до даних. Відсутність стандартів істотно обмежує потенційні можливості розподілених СКБД. Крім того, не існує інструментальних засобів і методологій, здатних допомогти користувачам в перетворенні централізованих систем в розподілені.

Недолік досвіду

Нині в експлуатації знаходиться вже декілька системи-прототипів і розподілених СКБД спеціального призначення, що дозволило уточнити вимоги до використовуваних протоколів і встановити коло основних проблем. Проте на поточний момент розподілені системи загального призначення ще не набули широкого поширення. Відповідно, ще не накопичений необхідний досвід промислової експлуатації розподілених систем, порівнюваний з досвідом експлуатації централізованих систем. Такий стан справ є серйозним стримуючим чинником для багатьох потенційних прибічників цієї технології.

Таблиця 15.1.

Переваги і недоліки розподілених СКБД

Переваги	Недоліки
Відображення структури організації	Підвищення складності
Разделяемость і локальна автономність	Збільшення вартості
Підвищення доступності даних	Проблеми захисту
Підвищення надійності	Ускладнення контролю за цілісністю даних
Підвищення продуктивності	Відсутність стандартів
Економічні вигоди	Недолік досвіду
Модульність системи	Ускладнення процедури розробки бази даних

Ускладнення процедури розробки бази даних

Розробка розподілених баз даних, окрім звичайних труднощів, пов'язаних з процесом проектування централізованих баз даних, вимагає прийняття рішення про фрагментацію даних, розподіл фрагментів по окремих сайтах і організації процедур реплікації даних. Усі ці проблеми детальніше будуть розглядатись нижче.

Усі переваги і недоліки, властиві розподіленим СКБД, перераховані в таблиці 15.1.

15.2. Гомогенні і гетерогенні розподілені СКБД

Розподілені СКБД можна класифікувати як **гомогенні** і **гетерогенні**. У гомогенних системах усі сайти використовують один і той же тип СКБД. У гетерогенних системах на сайтах можуть функціонувати різні типи СКБД, що використовують різні моделі даних, тобто гетерогенна система може включати сайти з реляційними, сітковими, ієрархічними або об'єктно-орієнтованими СКБД.

Гомогенні системи значно простіше проектувати і супроводжувати. Крім того, подібний підхід дозволяє поетапно нарощувати розміри системи, послідовно додаючи нові сайти до вже існуючої розподіленої системи. Додатково з'являється можливість підвищувати продуктивність системи за рахунок організації на різних сайтах паралельної обробки інформації.

Гетерогенні системи зазвичай виникають в тих випадках, коли незалежні сайти, що вже експлуатують свої власні системи з базами даних, інтегруються в новостворювану розподілену систему. У гетерогенних системах для організації взаємодії між різними типами СКБД потрібно буде організувати трансляцію передаваних повідомлень. Для забезпечення прозорості відносно типу використовуваної СКБД користувачі кожного з сайтів повинні мати можливість вводити запити, що цікавлять їх, на мові тієї СКБД, яка використовується на цьому сайті. Система повинна узяти на себе локалізацію необхідних даних і виконання трансляції передаваних повідомлень. У загальному випадку дані

можуть бути потрібні з друго-го сайту, який характеризується такими особливостями, як:

- інший тип використовуваного устаткування;
- інший тип використовуваної СКБД;
- інший тип використовуваного устаткування і СКБД.

Якщо використовується інший тип устаткування, проте на сайті встановлений той же самий тип СКБД, методи виконання трансляції цілком очевидні і включають заміну кодів і зміну довжини слова. Якщо типи використовуваних на сайтах СКБД різні, процедура трансляції ускладнюється тим, що необхідно відображувати структури даних однієї моделі у відповідні структури даних іншої моделі. Наприклад, відношення в реляційній моделі даних мають бути перетворені в записи і набори, типові для мережевої моделі даних. Крім того, потрібно буде транслювати текст запитів з однієї використовуваної мови на іншій (наприклад, запити SQL-оператор SELECT потрібно буде перетворити в запити з операторами FIND і GET мови маніпулювання даними сіткової СКБД). Якщо відрізняються і тип використовуваного устаткування, і тип програмного забезпечення, потрібно буде виконувати обидва види трансляції. Усе викладене вище назвичайно ускладнює обробку даних в гетерогенних СКБД.

Типове рішення, використовуване в деяких реляційних системах, полягає в тому, що окремі частини гетерогенних розподілених систем повинні використовувати шлюзи, призначені для перетворення мови і моделі даних кожного з використовуваних типів СКБД в мову і модель даних реляційної системи. Проте підходу з використанням шлюзів властиві деякі серйозні обмеження. По-перше, шлюзи не дозволяють організувати систему управління транзакціями навіть для окремих пар систем. Іншими словами, шлюз між двома системами представляє собою не більше, ніж транслятор запитів. Наприклад, шлюзи не дозволяють системі координувати управління паралельністю і процедурами відновлення транзакцій, що включають оновлення даних в обох базах. По-друге, використання шлюзів покликане лише вирішити завдання трансляції запитів з мови однієї СКБД на мову іншої СКБД. Тому вони не дозволяють впоратися з проблемами, викликаними неоднідністю структур і представленням даних в різних схемах.

Розробка розподілених реляційних баз даних

У попередніх заняттях вашій увазі була запропонована методологія концептуального і логічного проектування централізованих реляційних баз даних. На даному занятті ми розглянемо додаткові чинники, які повинні братися до уваги при розробці розподілених реляційних баз даних. Зокрема, обговоримо наступні аспекти проектування розподілених систем.

- **Фрагментація**. Будь-яке відношення може бути розділене на деяку кількість частин, які називаються фрагментами, які потім розподіляються по різних сайтах. Існують два основні типи фрагментів: горизонтальні і вертикальні. Горизонтальні фрагменти є підмножинами кортежів, а вертикальні – підмножини атрибутів.

– **Розподіл**. Кожен фрагмент зберігається на сайті, вибраному з врахуванням "оптимальної" схеми їх розміщення.

– **Реплікація**. СКРБД може підтримувати актуальну копію деякого фрагмента на декількох різних сайтах.

Розподіл даних

Існують чотири альтернативні стратегії розміщення даних в системі: централізоване, роздільне (фрагментоване), розміщення з повною реплікацією і з вибірковою реплікацією. Нижче ми дізнаємося, які результати дає використання кожної з цих стратегій, і розглянемо їх у світлі досягнення визначених вище цілей.

Централізоване розміщення

Ця стратегія передбачає створення на одному з сайтів єдиної бази даних під управлінням СКБД, доступ до якої матимуть усі користувачі мережі (ця стратегія під назвою "розподілена обробка" вже розглядалася нами вище.) В цьому випадку локальність посилань мінімальна для усіх сайтів, за виключенням центрального, оскільки для отримання будь-якого доступу до даних потрібна установка мережевого з'єднання. Відповідно рівень витрат на передачу даних буде високий. Рівень надійності і доступності в системі низький, оскільки відмова на центральному сайті викличе параліч роботи усієї системи.

Роздільне (фрагментоване) розміщення

В цьому випадку база даних розбивається на фрагменти, що не перетинаються, кожен з яких розміщується на одному з сайтів системи. Якщо елемент даних буде розміщений на тому сайті, на якому він найчастіше використовується, отриманий рівень локальності посилань буде високий. За відсутності реплікації вартість зберігання даних буде мінімальна, але при цьому буде невисокий також рівень надійності і доступності даних в системі. Проте він буде вищий, ніж в попередньому варіанті, оскільки відмова на будь-якому з сайтів викличе втрату доступу тільки до тієї частини даних, яка на ньому зберігалася. При правильно вибраному способі розподілу даних рівень продуктивності в системі буде відносно високим, а рівень затрат на передачу даних – низьким.

Розміщення з повною реплікацією

Ця стратегія передбачає розміщення повної копії усієї бази даних на кожному з сайтів системи.

Розміщення з вибірковою реплікацією

Ця стратегія є комбінацією методів фрагментації, реплікації і централізації.

Фрагментація. Призначення фрагментації

Коректність фрагментації

Фрагментація даних не повинна виконуватися непродумано, навмання. Існують три правила, яких слід обов'язково дотримуватися при проведенні фрагментації.

– Повнота. Якщо екземпляр відношення R розбивається на фрагменти, наприклад R_1, R_2,.., R_n, то кожен елемент даних, присутній відносно

R, має бути присутнім, принаймні, в одному із створених фрагментів. Виконання цього правила гарантує, що які-небудь дані не будуть втрачені в результаті виконання фрагментації.

Таблиця 15.2.

Порівняльні характеристики різних стратегій розподілу даних

Розміщення даних	Локальність посилань	Надійність і доступність	Продуктивність	Вартість пристроїв зберігання	Витрати на передачу даних
Централізоване	Найнижча	Найнижча	Незадовільна	Найнижча	Найвищі
Фрагментоване	Висока[1]	Низька для окремих елементів; висока для системи в цілому	Задовільна[1]	Найнижча	Низькі
Повна реплікація	Найвища	Найвища	Хороша для операцій читання	Найвища	Високі для операцій обновлення, низькі для операцій читання
Вибіркова реплікація	Висока[1]	Низька для окремих елементів, висока для системи	Задовільна [1]	Середня	Низькі

1 За умови якісного проектування.

– Відновлюваність. Повинна існувати операція реляційної алгебри, що дозволяє відновити відношення R з його фрагментів. Це правило гарантує збереження функціональних залежностей.

– Неперетинання. Якщо елемент даних d_i присутній у фрагменті R_i, то він не має бути одночасно присутнім в якому-небудь іншому фрагменті. Винятком з цього правила являється операція вертикальної фрагментації, оскільки в цьому випадку в кожному фрагменті мають бути присутніми атрибути первинного ключа, необхідні для відновлення початкового відношення. Це правило гарантує мінімальну надлишковість даних у фрагментах.

У разі горизонтальної фрагментації елементом даних являється кортеж, а у разі вертикальної фрагментації – атрибут.

Типи фрагментації

Існують два основні типи фрагментації: **горизонтальна** і **вертикальна.** Горизонтальні фрагменти є підмножинами кортежів відношення, а вертикальні – підмножини атрибутів відношення, як показано на рисунку 15.2.

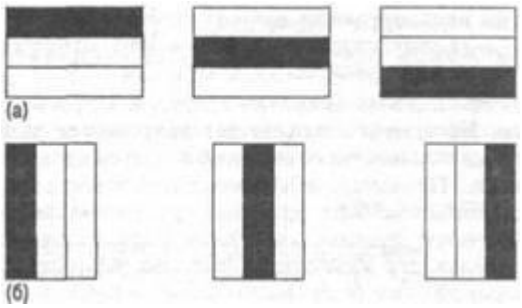


Рисунок 15.2. – Різні типи фрагментації: а) горизонтальна; б) вертикальна

Крім того, існують ще два типи фрагментації: змішана (рисунок 15.3) і похідна (що є варіантом горизонтальної фрагментації).

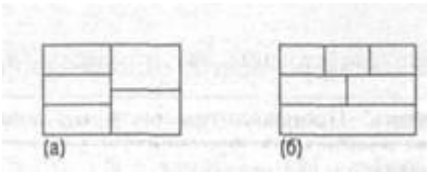


Рисунок 15.3.– Змішана фрагментація: а) горизонтально розділені вертикальні фрагменти; б) вертикально розділені горизонтальні фрагменти

Відмова від фрагментації

Останній варіант можливої стратегії полягає у відмові від фрагментації відношень. Наприклад, відношення Branch (відділення компанії) містить невелику кількість кортежів, котрі відносно рідко оновлюються. Замість того, щоб спробувати виконати горизонтальну фрагментацію цього відношення (наприклад, по номеру відділення компанії), має сенс залишити це відношення нефрагментованим і просто розмістити на кожному з сайтів його репліковані копії. Це перший етап типової процедури визначення схеми фрагментації (пошук відношень, які не вимагають фрагментації). Потім слід проаналізувати відношення, розташовані на одиничній стороні зв'язків типу "один до багатьох", і підібрати для них оптимальні схеми фрагментації. На останньому етапі аналізуються відношення, розміщені на множинній стороні тих же зв'язків. Саме вони найчастіше є кандидатами для застосування похідної фрагментації.

Забезпечення прозорості в РСКБД

Розподілені СКБД можуть забезпечувати різні рівні прозорості. Проте у будь-якому випадку переслідується одна і таже мета: зробити роботу з розподіленою базою даних абсолютно аналогічної роботі із звичайною централізованою СКБД. Виділимо чотири основні типи прозорості, які можуть мати місце в системі з розподіленою базою даних.

- Прозорість розподіленості.
- Прозорість транзакцій.
- Прозорість виконання.
- Прозорість використання СКБД.

Прозорість розподіленості

Прозорість розподіленості бази даних дозволяє кінцевим користувачам сприймати базу даних як єдине логічне ціле.

Прозорість транзакцій

Прозорість транзакцій в середовищі розподілених СКБД означає, що при виконанні будь-яких розподілених транзакцій гарантується збереження цілісності і узгодженості розподіленої бази даних. Розподілена транзакція здійснює доступ до даних, що зберігаються у більше, ніж одному місці розташування. Кожна з транзакцій розділяється на декілька субтранзакцій – по одній для кожного сайту, до даних якого здійснюється доступ. На віддалених сайтах субтранзакції представляються агентами.

СКРБД повинна гарантувати атомарність глобальних транзакцій, а це означає, що усі її субтранзакції будуть або зафіксовані, або скасовані. Таким чином, СКРБД повинна синхронізувати виконання глобальної транзакції так, щоб мати гарантії, що усі її субтранзакції були успішно завершені до того, як почалася фінальна операція фіксації результатів усієї глобальної транзакції.

Прозорість виконання

Прозорість виконання вимагає, щоб робота в середовищі СКРБД виконувалася точно так, як і в середовищі централізованої СКБД.

Обробник розподілених запитів виробляє стратегію виконання, котра є оптимальною з точки зору деякої вартісної функції. Зазвичай розподілені запити оцінюються за такими показниками:

– час доступу, що включає фізичний доступ до даних на диску;

– час роботи центрального процесора, що витрачається на обробку даних в первинній пам'яті;

– час, необхідний для передачі даних по мережевих з'єднаннях.

Розглянемо спрощений варіант реляційної схеми програми з базою даних, що включає наступних три відношення:

Property (Еш, City) 10 000 записів, що зберігаються в Лондоні.

Renter(Їм, Max_Price) 100 000 записів, що зберігаються в Глазго.

Viewing(Ешг, Rno) 1 000 000 записів, що зберігаються в Лондоні.

Щоб отримати список об'єктів нерухомості в Абердине, які були оглянуті потенційними покупцями, згідними придбати об'єкти нерухомості дорожче 200000, можна скористатися наступним SQL – запитом:

SELECT p.pno FROM property p INNER JOIN

((renter r INNER JOIN viewing v ON r.rno = v.rno)

ON p.pno = v.pno

WHERE p.city = 'Aberdeen' AND r.max price > 200000;

Для простоти припустимо, що кожен кортеж у відношеннях має довжину, рівну 100 символам, що існує тільки 10 покупців, згідних придбати нерухомість за ціною більше 200000, і що в місті Абердин було проведено 100000 оглядів об'єктів нерухомості. Вважатимемо також, що час виконання обчислень несуттєвий в порівнянні з часом передачі даних, а швидкість передачі даних по каналах зв'язку складає 10000 символів в секунду, причому на кожне

повідомлення, що відправляється між двома сайтами, припадає затримка доступу, рівна 1 секунді.

Проаналізовано шість можливих стратегій виконанняцього запиту і для кожного з них обраховано відповідний час реакції системи.

Стратегія 1. Переслати відношення Renter в Лондон і виконати там обробку запиту:
Час = 1+(100 000 * 100/10 000) = 16,7 хв.

Стратегія 2. Переслати стосунки Property і Viewing в Глазго і виконати там обробку запиту:
Час = 2+[(1 000 000 + 10 000) * 100/10 000] = 28 год.

Стратегія 3. З'єднати стосунки Property і Viewing в Лондоні, вибрати кор-тежи для об'єктів нерухомості, розташованих в Абердине, а потім для кожного з відібраних кортежів перевірити в Глазго, чи встановив цей клієнт значення стелі допустимої стои-мости нерухомості Max Price > 200 000 фунтів стерлінгів. Про-верка кожного кортежу припускає відправку двох повідомлень : запиту і відповіді.
Час = 100 000 * (1+100/10 000) + 100 000 * 1 = 2,3 дня.

Стратегія 4. Вибрати в Глазго кортежі клієнтів, що встановили значення Max Price > 200 000 фунтів стерлінгів, після чого для кожного з j їх перевірити в Лондоні, чи оглядав цей клієнт об'єкти не-движимости в Абердине. І в цьому випадку кожна перевірка вклю-чает відправку двох повідомлень.
Час = 10*(1+100/10 000) + 10* 1 ≈20 с.

Стратегія 5. З'єднати стосунки Property і Viewing в Лондоні, вибрати сведе-ния про огляди об'єктів в Абердине, виконати проекцію ре-зультирующей таблиці по атрибутах Рпо і Rno, після чого пере-слать отриманий результат в Глазго для відбору кортежів по ус-ловию Max Price > 200 000 фунтів стерлінгів. Для спрощення припустимо, що довжина кортежу після операції проекції також дорівнює 100 символам.
Час = 1 + (100 000 * 100/10 000) = 16,7 хв.

Стратегія 6. Вибрати клієнтів зі значенням атрибуту Max_Price > 200 000 фун-тов стерлінгів в Глазго і переслати результуючу таблицю в Лондон для відбору відомостей про огляди об'єктів нерухомості в р. Абердин:
Час = 1 + (10 * 100/10 000) ≈1с.

Прозорість використання СКБД

Прозорість використання СКБД дозволяє приховати від користувача СКРБД той факт, що на різних сайтах можуть функціонувати різні локальні СКБД. Тому цей тип прозорості застосовний тільки у разі гетерогенних розподілених систем. Як правило, це один з найскладніших в реалізації типів прозорості.

Декілька слів на закінчення

На початку цього розділу, присвяченого обговоренню різних типів прозорості в середовищі розподілених СКБД, відзначалося, що досягнення

повної прозорості не усіма розцінюється як бажана мета. Як ми вже бачили, концепція прозорості не може бути віднесена до типу "усе або нічого", оскільки може бути організована на декількох різних рівнях. Кожен з рівнів має на увазі наявність певного набору угод між сайтами, що входять в систему. Наприклад, при повній прозорості сайти повинні наслідувати загальну угоду з таких питань, як модель даних, інтерпретація схем, представлення даних і набір функціональності, що надається на кожному з сайтів. З іншого боку спектру знаходяться непрозорі системи, в яких єдиною угодою є формат обміну даними і набір функціональних можливостей, що надаються кожним сайтом в системі.

Таблиця 15.3.

Порівняння результатів застосування різних стратегій для обробки одного і того ж розподіленого запиту

№	Стратегія	Час
1	Переслати відношення Renter в Лондон і виконати там обробку запиту	16,7 хв.
2	Переслати стосунки Property і Viewing в Глазго і виконати там обробку запиту	28 ч.
3	З'єднати стосунки Property і Viewing в Лондоні, вибрати кортежі для об’єктів нерухомості в Абердині, а потім для кожного з відібраних кортежів провірити в Глазго, чи встановив цей клієнт максимальне значення допустимої вартості нерухомості Max Price > 200000	2,3 дні
4	Вибрати в місті Глазго кортежі клієнтів, що встановили значення Max Price > 200000, після чого для кожного з них перевірити в Лондоні, чи оглядав цей клієнт об'єкти нерухомості в Абердині	20 с.
5	З'єднати стосунки Property і Viewing в Лондоні, вибрати відомості осмотрах об'єктів в Абердине, виконати проекцію результуючої таблиці по атрибутах Рпо і Rno, після чого переслати отриманий результат в Глазго для відбору кортежів по умові Max Price > 200 000	16,7 хв.
6	Вибрати клієнтів зі значенням атрибуту Max Price > 200000 в Глазго і переслати результуючу таблицю в Лондон для відбору відомостей про огляди об'єктів нерухомості в Абердині	1с.

Дванадцять правив Дейта для РСКБД

Наостанок ми перерахуємо дванадцять правив (чи цілей), які були сформульовані Дейтом для типової СКРБД. Основою для побудови усіх цих правил є те, що розподілена СКБД повинна сприйматися кінцевим користувачем точно так, як і звична йому централізована СКБД. Дані правила схожі з дванадцятьма правилами Кодда для реляційних систем.

Правило 0. Основний принцип

З точки зору кінцевого користувача розподілена система повинна виглядати в точності так, як і звичайна, нерозподілена система.

Правило 1. Локальна автономність

Сайти в розподіленій системі мають бути автономними. У даному контексті автономність означає наступне:

– локальні дані належать локальним власникам і супроводжуються локально;

– усі локальні процеси залишаються чисто локальними;

– усі процеси на заданому сайті контролюються тільки цим сайтом.

Правило 2. Відсутність опори на центральний сайт

У системі не повинно бути жодного сайту, без якого система не зможе функціонувати. Це означає, що в системі не повинно існувати центральних серверів таких служб, як управління транзакціями, виявлення взаємних блокувань, оптимізація запитів і управління глобальним системним каталогом.

Правило 3. Безперервне функціонування

В ідеалі, в системі ніколи не повинна виникати потреба в плановій зупинці її функціонування для виконання таких операцій, як:

– додавання або видалення сайту з системи;

– динамічне створення або видалення фрагментів з одного або декількох сайтів.

Правило 4. Незалежність від розташування.

Незалежність від розташування еквівалентна прозорості розташування. Користувач повинен діставати доступ до бази даних з будь-якого з сайтів. Більше того, користувач повинен діставати доступ до будь-яких даних так, як якби вони зберігались на його сайті, незалежно від того, де вони фізично зберігаються.

Правило 5. Незалежність від фрагментації

Користувач повинен діставати доступ до даних незалежно від способу їх фрагментації.

Правило 6. Незалежність від реплікації

Користувач не повинен потребувати відомостей про наявність реплікації даних. Це означає, що користувач не матиме засобів для діставання прямого доступу до конкретної копії елементу даних, а також не повинен піклуватися про оновлення усіх наявних копій елементу даних.

Правило 7. Обробка розподілених запитів

Система повинна підтримувати обробку запитів, що посилаються до даних, розташованих на більш, ніж одному сайті.

Правило 8. Обробка розподілених транзакцій

Система повинна підтримувати виконання транзакцій, як одиниці відновлення. Система повинна гарантувати, що виконання як глобальних, так і локальних транзакцій відбуватиметься із збереженням чотирьох основних властивостей транзакцій, а саме: атомарності, узгодженості, ізольованості і тривалості.

Правило 9. Незалежність від типу устаткування

СКРБД має бути здатна функціонувати на устаткуванні з різними обчислювальними платформами.

Правило 10. Незалежність від операційної системи

Прямим наслідком попереднього правила є вимога, згідно з яким СКРБД має бути здатна функціонувати під управлінням різних операційних систем.

Правило 11. Незалежність від мережевої архітектури

СКРБД має бути здатна функціонувати в мережах з різною архітектурою і типами носія.

Правило 12. Незалежність від типу СКБД

СКРБД має бути здатна функціонувати поверх різних локальних СКБД, можливо, з різним типом використовуваної моделі даних. Іншими словами, СКРБД повинна підтримувати гетерогенність.

Останніх чотири правила є доки лише ідеальними. Оскільки їх формулювання є занадто загальним, і внаслідок недостатності наявних стандартів на комп'ютерну і мережеву архітектуру, в осяжному майбутньому можна чекати лише часткового задоволення розробниками вказаних вимог.

Резюме

– Розподілена база даних є набором логічно пов'язаних між собою даних (і їх описів), що розділяються, які фізично розподілені в деякій комп'ютерній мережі. СКРБД є програмний комплексом, призначеним для прозорого управління розподіленої базою даних.

– Розподілену СКБД не слід змішувати з розподіленою обробкою, при якій доступ до централізованої СКБД одночасно надається багатьом користувачам в комп'ютерній мережі. СКРБД також відрізняється від паралельної СКБД, в якій локальна система управління базою даних функціонує з використанням декількох процесорів і пристроїв вторинної пам'яті, що дозволяє організувати паралельне виконання операцій (якщо це можливо) з метою підвищення продуктивності системи.

– Переваги СКРБД полягають в тому, що вона дозволяє відбити структуру організації і підвищує можливості спільного використання видалених даних, підвищує надійність, доступність і продуктивність системи, дозволяє отримати економію коштів, а також організувати модульне збільшення розмірів системи.

– Усі взаємодії виконуються за допомогою мережевих з'єднань, які можуть бути як локальними, так і глобальними. Локальні мережеві з'єднання встановлюються на невеликі відстані, але забезпечують велику пропускну спроможність, чим глобальні.

– Аналогічно тому, як централізована СКБД повинна надавати опреде-ленный набір стандартних функцій, розподілена СКБД повинна надавати розширені можливості установки з'єднань, включати розширену службу системного каталогу, забезпечувати розподілену обробку запитів, підтримувати розширені засоби розпаралелювання операцій, а також мати власну службу відновлення.

– Кожне відношення може бути розділене на деяку кількість частин, званих фрагментами. Фрагменти можуть бути горизонтальними, вертикальними, змішаними і похідними. Фрагменти розподіляються на одному або декількох сайтах. З метою поліпшення доступності даних і підвищення продуктивності системи для окремих фрагментів може бути організована реплікація.

– Визначення і розподіл фрагментів виконуються для досягнення наступних цілей: забезпечення локальності посилань, підвищення надійності і

доступності даних, забезпечення прийнятного рівня продуктивності, досягнення балансу між вартістю і місткістю пристроїв вторинної пам'яті, а також мінімізації витрат на передачу даних. Три основних пра-вила коректності фрагментації включають вимоги повноти, відновлюваності і непересікальності.

– Існують чотири стратегії розподіли, що визначають спосіб розміщення даних: централізований (єдина централізована база даних), фрагментований розподіл (кожен фрагмент розміщується на одному з сайтів), розподіл з повною реплікацією (повна копія усієї бази даних підтримується на кожному сайті) і розподіл з вибірковою репликацией (комбінація перших трьох способів).

– З точки зору користувача, СКРБД повинна виглядати точно так, як і звичайна централізована СКБД, що досягається за рахунок забезпечення различ-ных типів прозорості. Завдяки прозорості розподілу користувачі не потребують яких-небудь відомостей про існуючу в системі фрагментації/реплікацію. Прозорість транзакцій забезпечує збереження узгодженості глобальної бази навіть за наявності паралельного доступу до неї з боку безлічі користувачів і наявності в системі різних відмов. Прозорість виконання дозволяє системі ефективно обробляти запити, що включають звернення до даних на декількох сайтах. Прозорість використання СКБД дозволяє системі функціонувати поверх установлен-ных на окремих сайтах локальних СКБД різного типу.

Керування розподіленою паралельністю.

– Цілі, що стояти при обробці розподілених транзакцій, не відрізняються від переслідуваних при обробці транзакцій у централізованих системах. Однак досягти цих цілей у випадку СКРБД складніше, оскільки потрібно забезпечити неподільність не тільки глобальної транзакції в цілому, але й всіх її субтранзакцій.

– Якщо графіки виконання транзакцій на шкірному із сайтів упорядковані, то глобальний графік (об' єднання всіх локальних графіків) також буде впорядкований з тим же типом, що й локальні графіки. Це означає, що всі субтранзакції будуть з'являтися в тому самому порядку в еквівалентних послідовних графіках на всіх сайтах.

– Для забезпечення розподіленої впорядкованості можуть використовуватися два методи: з використанням блокування або тимчасових міток. Двофазний протокол блокування вимагає, щоб у транзакції всі необхідні блокування булі встановлені до скасування хоча б однієї з них. Протоколи двофазного блокування можуть використати централізовану схему, схему з первинними копіями й розподіленою схемою. Крім того, може використатися метод блокування більшості. При використанні тимчасових міток транзакції впорядковуються таким чином, що найбільш старі транзакції у випадку конфлікту одержують перевагу.

– Виявлення ситуацій розподіленого взаємного блокування вимагає злиття локальних графів очікування з наступним аналізом глобального графа на наявність петель. При виявленні петлі для її усунення одна або більше транзакцій повинні бути скасовані й перезапущені. У розподілених СКБД

існують три загальні схеми виявлення розподіленого взаємного блокування: централізована, ієрархічна й розподілена.

– Специфічними причинами відмов у розподіленому середовищі є втрата повідомлень, відмова ліній зв'язку й мережних з' єднань, відмова окремих сайтів і розчленовування мережі. Для організації відновлення на шкірному із сайтів створюється локальний журнал реєстрації подій, що використовується для відкату й повторного виконання транзакцій, необхідного при відновленні після відмови.

– Двофазний протокол фіксації транзакцій складається з етапів голосування й ухвалення глобального рішення. На першому етапі координатор опитує учасників про їхню готовність до фіксації транзакції. Якщо хоча б один з учасників проголосує за скасування транзакції, глобальна транзакція й всі її локальні субтранзакції будуть скасовані. Глобальна транзакція може бути зафіксована тільки в тому випадку, якщо всі учасники проголосували за фіксацію результатів. При використанні двофазного протоколу фіксації транзакцій у середовищі, підданої відмовам, деякі сайти можуть залишитися заблокованими.

– Трифазний протокол фіксації транзакцій неблокуючий. Він припускає розсилання координатором транзакції між етапами голосування й ухвалення рішення додаткових повідомлень на адресі всіх учасників транзакції. Ці повідомлення вказують учасникам на необхідність перейти в стан предфіксації транзакції.

– Модель X/Open DTP являє собою один з варіантів архітектури обробки розподілених транзакцій, побудованої на застосуванні протоколу OSI – TP і двофазного протоколу фіксації транзакцій. У цій моделі визначаються прикладні програмні інтерфейси й методи взаємодії між додатками, що запускають транзакції, менеджерами транзакцій, менеджерами ресурсів і менеджерами передачі даних.

– Реплікацією називається процес генерації й відтворення декількох копій даних на одному або більше сайтів. Це дуже важливий механізм, оскільки з його допомогою організації можуть надавати своїм користувачам доступ до актуальної інформації там і тоді, коли це їм потрібно. Використання реплікації дозволяє досягти багатьох переваг, включаючи підвищення продуктивності (у тихий випадках, коли централізовані ресурси виявляються перевантаженими), підвищення надійності зберігання й доступності даних, а також забезпечення підтримки мобільних користувачів і сховищ даних, призначених для систем підтримки прийняття рішень.

ЛЕКЦІЯ 16. КЕРУВАННЯ РОЗПОДІЛЕНОЮ ПАРАЛЕЛЬНІСТЮ

16.1 Управління розподіленими транзакціями

Раніше було відмічено, що цілі розподіленої обробки транзакцій аналогічні тим, які переслідуються при обробці транзакцій в централізованих системах, хоча використовувані для цього методи істотно складніші, оскільки розподілена СКБД повинна забезпечувати нерозривність усієї глобальної транзакції, а також кожній з субтранзакцій, що входять до її складу. **Диспетчер транзакцій** координує виконання транзакцій, що запускаються прикладними програмами, і взаємодіє з **планувальником**, відповідальним за реалізацію вибраної стратегії управління паралельним виконанням в системі. Мета роботи планувальника полягає в досягненні максимального рівня розпаралелювання в системі з одночасною гарантією відсутності якого-небудь впливу транзакцій, що паралельно виконуються, одна на одну, що необхідно для постійного збереження бази даних в узгодженому стані. Якщо в процесі виконання транзакції відбувається відмова, **диспетчер відновлення** забезпечує відновлення бази даних і переведення її в узгоджений стан, який вона мала до початку виконання транзакції. Диспетчер відновлення також відповідає за відновлення бази даних до деякого узгодженого стану і після відмов системи. **Диспетчер буферів** забезпечує передачу даних між дисковими пристроями і оперативною пам'яттю комп'ютера.

У розподіленій СКБД усі ці модулі як і раніше є в кожній локальній СКБД. Крім того, на кожному вузлі функціонує диспетчер глобальних транзакцій, або координатор транзакцій, що координує виконання глобальних і локальних транзакцій, ініційованих на цьому вузлі. Усі взаємодії між вузлами здійснюються через компонент передачі даних (диспетчери транзакцій на різних вузлах не взаємодіють безпосередньо один з одним).

Процедура виконання глобальної транзакції, запущеної на вузлі здійснюється таким чином.

– Координатор транзакцій ($ТС_1$) на вузлі S_1, ділить транзакцію на декілька субтранзакцій, виходячи з інформації, що зберігається в глобальному каталозі системи.

– Компонент передачі даних на вузлі – відправляє опис відповідних субтранзакцій на вузли S_2 і S_3.

– Координатори транзакцій на вузлах S_2 і S_3 організовують виконання субтранзакцій, що поступили. Результати їх виконання передаються координаторові $ТС_1$ за допомогою компонентів передачі даних. Увесь описаний процес схематично представлений на рисунку 16.1.

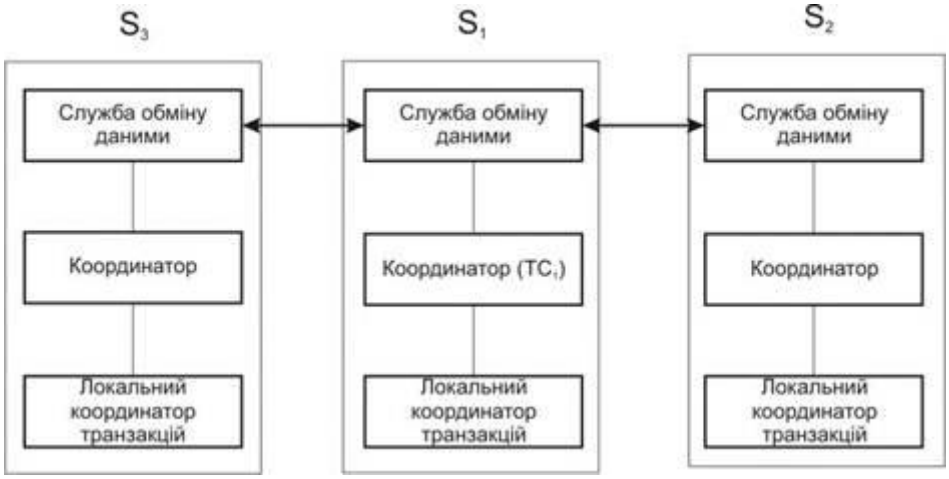


Рисунок 16.1. – Схема координації виконання розподілених транзакцій

Грунтуючись на приведеній вище загальній схемі обробки розподілених транзакцій, можна перейти до опису протоколів управління паралельним виконанням, виявлення взаимоблокировок і відновлення.

16.2. Управління паралельним виконанням в розподіленому середовищі

У цьому розділі будуть представлені протоколи, які можуть використовуватися для організації управління паралельним виконанням в розподілених СКБД. Але спочатку слід ознайомитися з цілями, які стоять перед механізмом управління паралельним виконанням в розподіленому середовищі.

Цілі управління паралельним виконанням в розподіленому середовищі

Якщо припустити, що в системі не існує відмов, то призначення механізму управління паралельним виконанням полягатиме в забезпеченні узгодженості елементів даних і завершенні кожної елементарної операції в межах деякого встановленого проміжку часу. Крім того, прийнятний механізм управління паралельним виконанням в розподіленій СКБД повинен забезпечувати наступне:

– стійкість до відмов на вузлі і в лініях зв'язку;

– високий рівень паралельності, що задовольняє існуючим вимогам продуктивності;

– невисокий додатковий рівень споживання процесорного часу і інших системних ресурсів;

– задовільні показники роботи з мережевими з'єднаннями, що мають велику тривалість часу затримки з'єднання;

– відсутність додаткових обмежень на структуру елементарних операцій.

У попередніх заняттях були описані основні проблеми, які можуть виникати, коли декілька користувачів мають одночасний доступ до бази даних. До них відносяться проблеми втраченого оновлення, залежності від проміжних результатів і неузгодженості обробки. Усі ці проблеми мають також місце і в розподіленому середовищі, але тут доводиться долати ще і труднощі, викликані

розподіленим зберіганням даних. Одна з них називається проблемою узгодженості багатьох копій даних і виникає в тих випадках, коли існує декілька копій одного елементу даних, розміщених в різних місцях. Очевидно, що для підтримки узгодженості глобальної бази даних при оновленні копійованого елементу даних на одному з вузлів необхідно відбити цю зміну і в усіх інших копіях цього елементу. Якщо оновлення не буде відбито в усіх копіях, база даних перейде в неузгоджений стан. У цьому розділі передбачається, що оновлення копійованих елементів виконується в системі синхронно, як частина транзакції, що включає початкову операцію оновлення.

Проблема впордкування субтранзакцій в розподіленому середовищі

Концепція впорядкування, описана раніше, може бути розширена з урахуванням особливостей зберігання даних в розподіленому середовищі. Якщо графік виконання транзакцій на кожному з вузлів є упорядковуваним, то глобальний графік (є об'єднанням усіх локальних графіків) також буде впорядковуваним, за умови, що послідовності локального впорядкування є ідентичними. Для цього необхідно, щоб усі субтранзакції розташовувалися в одному і тому ж порядку в еквівалентному послідовному графіку на усіх вузлах. Тому, якщо субтранзакцію Ti на вузлі S_1 позначити як, необхідно забезпечити, що якщо < то < для усіх вузлів S_x, на яких транзакції < мають субтранзакції.

Рішення по організації управління паралельним виконанням в розподіленому середовищі засновані на підходах з використанням механізму блокувань і часових відміток, які вже розглядалися стосовно централізованих баз даних. Тому, якщо дана множина транзакцій, які повинні виконуватися паралельно, то можуть розглядатися наступні два варіанти.

– Механізм блокування забезпечує, що графік паралельного виконання транзакцій буде еквівалентний деякому (непередбачуваному) варіанту послідовного виконання цих транзакцій.

– Механізм обробки часових відміток гарантує, що графік паралельного виконання транзакцій буде еквівалентний конкретному варіанту послідовного виконання цих транзакцій відповідно до їх часових відміток.

Якщо база даних являється або централізованою, або фрагментовано, але не використовує реплікації даних, то в ній існує тільки одна копія кожного елементу даних. Тому усі виконувані в ній транзакції являються або локальними, або можуть бути виконані на одному з віддалених вузлів. В цьому випадку можуть використовуватися протоколи для централізованих баз даних. Але ці протоколи мають бути доповнені, якщо в системі застосовується реплікація даних або виконуються транзакції, що включають доступ до елементів даних, розміщених на декількох вузлах. Крім того, у разі застосування протоколів, що використовують механізм блокувань, додатково слід гарантувати усунення взаємоблокувань в системі. Для цього буде потрібно виявлення взаємоблокувань не лише на кожному з локальних рівнів, але і на глобальному рівні, якщо взаємоблокування виникає при зверненні до даних, розміщених на декількох вузлах.

Протоколи блокування

Розглянемо наступні протоколи, засновані на двофазному блокуванні (2PL), які можуть застосовуватися для забезпечення впорядкування графіків в розподілених СКБД. До них відносяться централізований протокол двофазного блокування, двофазне блокування з первинними копіями, розподілений протокол двофазного блокування і блокування більшості копій.

Централізований протокол двофазного блокування

При використанні цього протоколу існує єдиний вузол, на якому зберігається уся інформація про блокування елементів даних в системі. Тому в усій розподіленій СКБД існує тільки один планувальник, або диспетчер блокувань, здатний встановлювати і знімати блокування з елементів даних. При запуску глобальної транзакції на вузлі S_1 централізований протокол двофазного блокування працює таким чином.

1. Координатор транзакцій на вузлі S_1 розділяє глобальну транзакцію на декілька субтранзакцій, використовуючи інформацію, що зберігається в глобальному системному каталозі. Координатор відповідає за дотримання узгодженості бази даних. Якщо транзакція передбачає оновлення копійованого елементу даних, координатор повинен забезпечити оновлення усіх існуючих копій цього елементу даних. Тому координатор повинен зажадати встановити виняткові блокування на усіх копіях оновлюваного елементу даних, а потім звільнити ці блокування. Для читання оновлюваного елементу даних координатор може вибрати будь-яку з існуючих копій. Зазвичай прочитується локальна копія, якщо така існує.

2. Локальні диспетчери транзакцій, які беруть участь у виконанні глобальної транзакції, запрошують і звільняють блокування елементів даних під управлінням центрального диспетчера блокувань, керуючись при цьому звичайними правилами протоколу двофазного блокування.

3. Центральний диспетчер блокувань перевіряє допустимість запитів, що поступають, на блокування елементів даних з урахуванням поточного стану блокування цих елементів. Якщо блокування є допустимим, диспетчер блокувань направляє на початковий вузол повідомлення, що повідомляє, що необхідне блокування елементу даних надане. Інакше запит поміщається в чергу, де і знаходиться аж до того моменту, коли необхідне блокування може бути надане.

Варіантом цієї схеми є випадок, коли координатор транзакцій виконує усі запити на блокування від імені локальних диспетчерів транзакцій. В цьому випадку диспетчер блокувань взаємодіє тільки з координатором транзакції, а не з окремими локальними диспетчерами транзакцій.

Перевага централізованого протоколу двофазного блокування полягає в тому, що його можна відносно просто реалізувати. Виявлення взаимоблокировок може виконуватися тими ж методами, що і в централізованих СКБД, оскільки уся інформація про блокування елементів знаходиться у розпорядженні єдиного диспетчера блокувань. Основний недолік цієї схеми пов'язаний з тим, що будь-яка централізація в розподіленій СКБД автоматично призводить до появи вузьких місць в системі і різко знижує рівень

її надійності і стійкості. Оскільки усі запити на блокування елементів даних прямують на єдиний центральний вузол, його швидкодія обмежує можливості усієї системи. Крім того, відмова цього вузла викликає порушення роботи усієї розподіленої системи, тому вона стає менш надійною. Проте цій схемі властивий відносно невисокий рівень витрат на передачу даних. Наприклад, глобальна операція оновлення за участю агентів (субтранзакцій) на вузлах за наявності центрального диспетчера блокувань може потребувати відправки йому не менше $2n + 3$ повідомлень, у тому числі:

- один запит на блокування;
- одне повідомлення про надання блокування;
- n сполучень з вимогою оновлення;
- n підтверджень про виконане оновлення;
- один запит на зняття блокування.

Двофазне блокування з первинними копіями

У цьому варіанті протоколу спроба подолати недоліки централізованого протоколу двофазного блокування робиться за рахунок розподілу функцій диспетчера блокувань по декількох вузлах. В даному випадку кожен локальний диспетчер відповідає за управління блокуванням деякого набору елементів даних. В процесі реплікації для кожного копійованого елементу даних одна з копій вибирається як первинна копія (primary copy), а усі інші розглядаються як вторинні (slave copy). Вибір первинного вузла може здійснюватися за різними правилами, причому вузол, який вибраний для управління блокуванням первинної копії даних, не обов'язково повинен містити саму цю копію.

Цей протокол є простим розширенням централізованого протоколу двофазного блокування. Основна відмінність полягає в тому, що при оновленні елементу даних координатор транзакцій повинен визначити, де знаходиться його первинна копія, і послати запит на блокування елементу відповідному диспетчерові блокувань. При оновленні елементу даних досить встановити виняткове блокування тільки його первинної копії. Після того, як первинна копія буде оновлена, внесені зміни можуть бути поширені на усі вторинні копії. Це поширення має бути виконане з максимально можливою швидкістю, щоб запобігти читанню іншими транзакціями застарілих значень даних. Проте немає необхідності виконувати усі оновлення у вигляді однієї елементарної операції. Цей протокол гарантує актуальність значень тільки первинної копії даних.

Подібний підхід може використовуватися в тих випадках, коли дані піддаються вибірковій реплікації, їх оновлення відбувається відносно рідко, а вузли не потребують використання новітньої копії усіх елементів даних. Недоліками цього підходу є ускладнення методів виявлення взаємоблокувань у зв'язку з наявністю декількох диспетчерів блокувань, а також збереження в системі певної міри централізації, оскільки запити на блокування первинної копії елементу можуть бути виконані тільки на єдиному вузлі. Останній недолік може бути частково компенсований за рахунок застосування резервних вузлів, що містять копію інформації про блокування елементів. Цей протокол характеризується меншими витратами на передачу повідомлень і вищим рівнем

продуктивності, чим централізований протокол двофазного блокування, в основному за рахунок менш частого використання віддалених блокувань.

Розподілений протокол двофазного блокування

У цьому протоколі також робиться спроба подолати недоліки, властиві централізованому протоколу двофазного блокування, але вже за рахунок розміщення диспетчерів блокувань на кожному вузлі системи. В даному випадку кожен диспетчер блокувань відповідає за управління блокуванням даних, таких, що знаходяться на його вузлі. Якщо дані не піддаються реплікації, цей протокол функціонує аналогічно протоколу двофазного блокування з первинними копіями. Інакше розподілений протокол двофазного блокування використовує особливий протокол управління реплікацією, що дістав назву "Читання однієї копії і оновлення усіх копій" (Read – One – Write – All – ROWA). В цьому випадку для операцій читання може використовуватися будь-яка копія копійованого елементу, але перш ніж можна буде відновити значення елементу, мають бути встановлені виняткові блокування на усіх копіях. У такій схемі управління блокуваннями здійснюється децентралізованим способом, що дозволить позбавитися від недоліків, властивих централізованому управлінню. Проте цьому підходу властиві свої недоліки, пов'язані з істотним ускладненням методів виявлення взаємоблокувань (через наявності багатьох диспетчерів блокувань) і зростанням витрат на передачу даних (в порівнянні з протоколом двофазного блокування з первинними копіями), які викликані необхідністю блокувати усі копії кожного оновлюваного елементу. У цьому протоколі виконання глобальної операції оновлення, що має агентів на n вузлах, зажадає передачі не менше $5n$ повідомлень, у тому числі:

- n сполучень із запитами на блокування;
- n сполучень з наданням блокування;
- n сполучень з вимогою оновлення елементу;
- n сполучень з підтвердженням виконаного оновлення;
- n сполучень із запитами на зняття блокування.

Це кількість повідомлень може бути скорочена до $4n$, якщо не передавати запити на зняття блокування, яке в цьому випадку виконуватиметься при обробці операції остаточної фіксації розподіленої транзакції.

Блокування більшості копій

Цей протокол можна вважати розширенням розподіленого протоколу двофазного блокування, в якому усувається необхідність блокування усіх копій реплікованого елементу даних перед його оновленням. В цьому випадку диспетчер блокувань також є на кожному з вузлів системи, де він управляє блокуваннями усіх даних, що розміщуються на цьому вузлі. Коли транзакції вимагається прочитати або записати елемент даних, копії якого є на n вузлах системи, вона повинна відправити запит на блокування цього елементу більше, ніж на половину з усіх тих n вузлів, де є його копії. Транзакція не має права продовжувати своє виконання, поки не встановить блокування на більшості копій елементу даних. Якщо їй не вдасться це зробити за деякий встановлений проміжок часу, вона відміняє свої запити і інформує усі вузли про відміну її виконання. Якщо більшість підтверджень будуть отримані, усі вузли

інформуються про те, що необхідний рівень блокування досягнутий. Блокування, що розділяється, на більшості копій може бути встановлене одночасно для будь-якої кількості транзакцій, а виняткове блокування на більшості копій може бути встановлене тільки для однієї транзакції.

В цьому випадку також усуваються недоліки, властиві централізованому підходу. Але цьому протоколу властиві власні недоліки, що полягають в підвищеній складності алгоритму, ускладненні процедур виявлення взаємоблокувань, а також необхідності відправки великої кількості повідомлень із запитами на встановлення блокування та з запитами на відміну блокування. Метод успішно працює, але показує себе надмірно жорстким відносно блокувань, що розділяються. Для читання досить заблокувати тільки одну копію елементу даних, а цей метод вимагає установки блокувань на більшості копій.

Протоколи з часовими відмітками

Протоколи для централізованих баз даних, що використовують часові відмітки, описані раніше. Завданням подібних протоколів є глобальне впорядкування транзакцій таким чином, що старіші транзакції (що мають меншу тимчасову відмітку) у разі конфлікту отримують пріоритет. У розподіленому середовищі необхідно також виробляти унікальні значення часових відміток, причому як локально, так і глобально. Очевидно, що використання на кожному вузлі системного годинника або накопичуваного лічильника подій вже не є прийнятним рішенням. Годинник на кожному вузлі може бути недостатньо синхронізований, а при використанні лічильників подій ніщо не перешкоджає виробленню одних і тих же значень лічильника одночасно на різних вузлах.

Загальним підходом в розподілених СКБД є конкатенація локальної часової відмітки з унікальним ідентифікатором вузла у форматі <локалъная_відмітка, ідентифікатор_вузла>. Значення ідентифікатора вузла має менший ваговий коефіцієнт, що гарантує впорядкування подій відповідно до моменту їх виникнення і лише потім відповідно до місця їх появи. Щоб запобігти виробленню більш завантаженими вузлами великих значень часових відміток в порівнянні з недовантаженими вузлами, необхідно використовувати певний механізм синхронізації значень часових відміток між вузлами. Кожен вузол поміщає свою поточну часову відмітку в повідомлення, що передаються на інші вузли. При отриманні повідомлення вузол-одержувач порівнює поточне значення його часової відмітки з отриманим і, якщо його поточна часова відмітка виявляється менше, міняє її значення на деяке інше, що перевищує те значення часової відмітки, яке було отримано ним в повідомленні. Наприклад, якщо вузол 1 з поточною часовою відміткою <10, 1> передає повідомлення на вузол 2 з поточною часовою відміткою <15,2>, то вузол 2 не змінює свою часову відмітку. І навпаки, якщо поточна часова відмітка на вузлі 2 рівна <5, 2>, то вузол 2 повинен змінити свою часову відмітку на <11, 2>.

Технології ActiveX та CORBA для побудови розподілених систем

ActiveX/DCOM – це корпоративна технологія з фірмовим знаком Microsoft. Головне її призначення – полегшення написання мережевих

розподілених об'єктно-орієнтованих застосувань. По суті, ActiveX – це ні що інше, як звичайний набір бібліотек, що значно полегшують процес кодування. Іноді запитують, в чому різниця між OLE і його складеними механізмами (OLE Automation, OLE Documents, OLE Controls) і елементами ActiveX? Відповідь дуже проста. OLE-механізми засновані на компонентній об'єктній моделі (COM, Component Object Model), яка дозволяє створювати об'єктно-орієнтовані застосування на одній машині у рамках операційної системи Windows, що функціонує на ній, а ActiveX використовує DCOM (Distributed Component Object Model) – розподілену компонентну об'єктну модель, яку реалізують бібліотеки ActiveX, що являються за об'ємом значно менше, чим бібліотеки OLE, а за швидкістю – швидше. Іншими словами, бібліотеки OLE переписані так, щоб забезпечувати функціональність, достатню для написання мережевих застосувань. Збереглася і сумісність – будь-який програмний компонент OLE працюватиме з бібліотеками ActiveX. Моделі COM і DCOM спираються на базові мережеві протоколи, такі як TCP/IP, і входять до складу операційної системи.

Що таке COM і DCOM

COM (Component Object Model) розшифровується просто – компонентна об'єктна модель. DCOM (Distributed Component Object Model) – це, відповідно, розподілена компонентна об'єктна модель.

Microsoft розробляє і підтримує DCOM виключно для створення серверів застосувань і управління розподіленими програмними об'єктами. На противагу Microsoft, два інших комп'ютерних гіганта – Sun і IBM – спираються на міжнародний стандарт CORBA. За бажання можна спільно використати обидві ці технології.

DCOM просто розширює можливості COM в частині реєстрації віддалених об'єктів, тому його іноді називають "COM з подовженим дротом".

Прикладні інтерфейси COM API – це усе ті ж, усім відомі OLE-інтерфейси, що ведуть своє походження від DDE (Dynamic Data Exchange), – динамічного обміну даними, що дозволяє єдиним способом в Windows здійснювати обмін інформацією між процесами.

По суті своїй COM є простим об'єктним розширенням OLE-технології, коли в локальній моделі засобів автоматизації OLE команди від клієнта проходять через модуль-ініціалізатор (proxy), а повідомлення – в модуль-перехідник (stub), де воно розбирається, аналізується і звідки посилається команда на сервер.

Наочніше ця ж схема, але вже в специфікації COM, представлена на рис. 16.2.

Модель COM значно простіша з точки зору програмістів і її набагато зручніше використати в невеликих локальних мережах.

У дистанційній моделі, яка тепер замість COM дістала назву DCOM, ініціалізатор і перехідник стали абсолютно іншими, тепер в них використовуються процедури дистанційного виклику (RPC) для передачі запитів клієнта по мережі до модуля Remote Server Process, що функціонує на сервері. Клієнтська частина DCOM, аналогічна колишньому механізму

Automation Manager, передає дані клієнта вказаному OLE-серверу і дані у відповідь по мережі до програми-клієнта, як показано на рис. 16.2.

Як можна бачити з приведених рисунків, чудовою властивістю DCOM є його прозорість як для користувача, так і для програміста. Можна створити і запустити на своїй машині сервер OLE, а потім пристосувати його для роботи в дистанційному режимі простим перенесенням OLE-сервера на віддалений ПК, на якому працює модуль адміністратор автоматизованого управління DCOM. Потім треба просто зареєструвати нове місце розташування OLE-сервера за допомогою звичайної утиліти (наприклад, з арсеналу засобів Visual Basic). Програмістові не доводиться міняти жодного рядка тексту програми, щоб використати розроблений OLE-сервер віддалено. Зараз майже у будь-якій RAD-системі є шаблони для створення OLE -серверів, методи і об'єкти яких доступні при роботі в дистанційному режимі. І, оскільки реєстрацію віддаленого сервера легко включити в процедуру установки програми-клієнта, не доводиться робити яких-небудь додаткових дій, щоб скористатися новими можливостями.

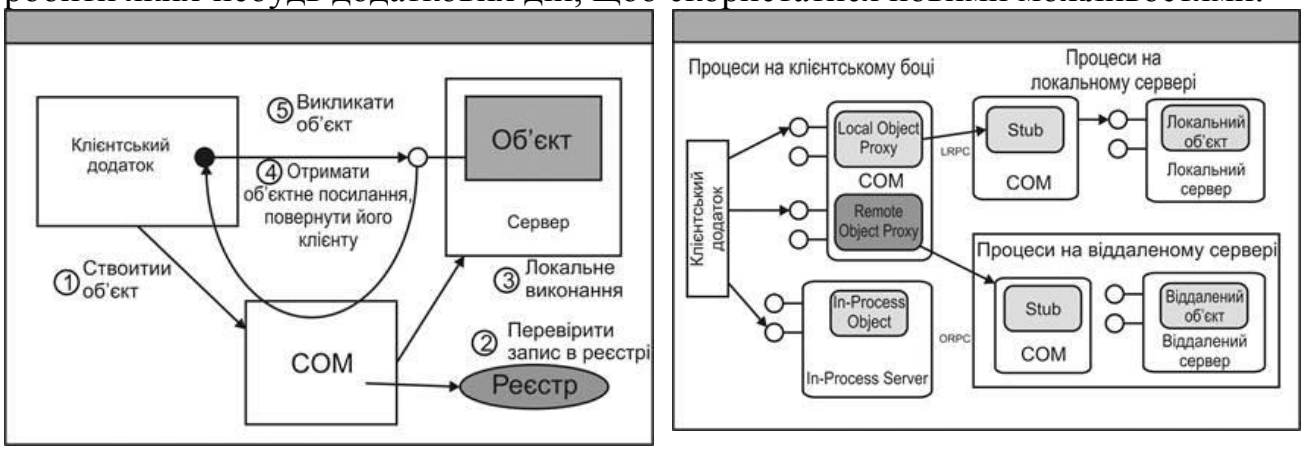


Рисунок 16.2. – Схема взаємодії для локального і віддаленого варіантів

Робота серверів OLE в дистанційному режимі має ряд істотних переваг, у тому числі спрощене адміністрування і обслуговування, можливість його багатократного використання будь-хто програмою-клієнтом OLE, багаторівнева система захисту доступу і достовірності даних. Але найголовніше перевага – це здатність віддалених OLE-серверів виконувати роль серверів застосувань, тобто своєрідних виконавців алгоритмів (у новій термінології – бізнес-правил) для доступу до даних в розподілених прикладних системах.

Реалізація цієї можливості є ключовим моментом в створенні трирівневої архітектури ділових застосувань Microsoft, в якій частина призначеного для користувача інтерфейсу розміщена на робочій станції клієнта, бізнес-логіка – на сервері середнього рівня, а надпотужні засоби обробки даних – на SQL-сервері бази даних (рис. 16.3).

Наприклад, трирівнева модель абсолютно не потрібна (і не обов'язково намагатися ділити програму на частини), якщо вона не вирішує загальну проблему ефективності.

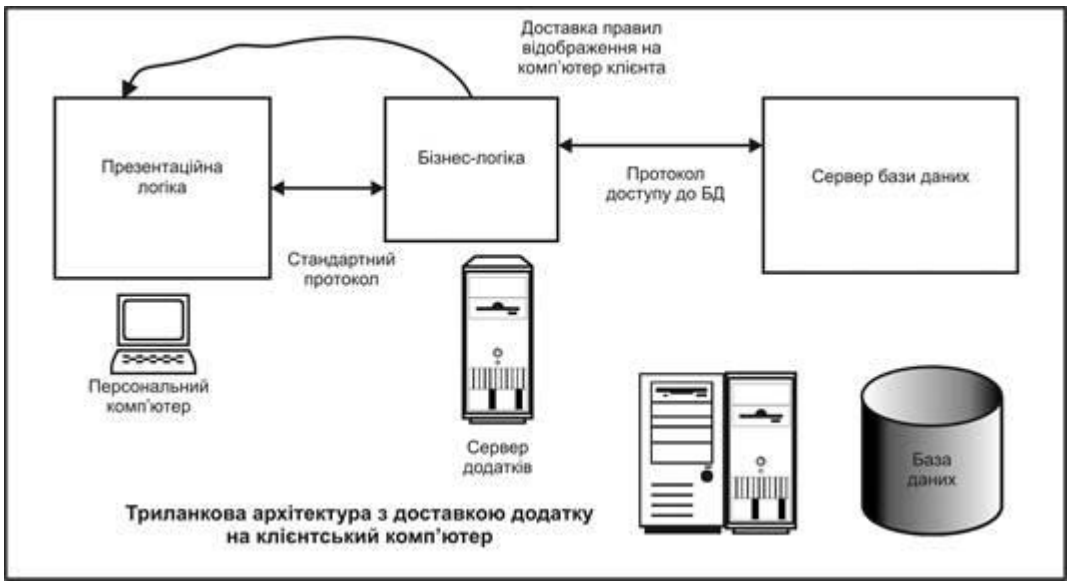


Рисунок 16.3. – Трирівнева архітектура бізнес-застосувань

Проте у фірмовій документації Microsoft, присвяченій OLE-об’єктам, стверджується: "Найчастіше розміщення сервер-об’єктів доступу до даних на тій машині, де знаходиться база даних, може істотно скоротити завантаження мережі і збільшити загальну швидкість передачі даних". Є приклади реалізації трирівневих клієнт-серверних застосувань, де результати перевершують усі очікування. Продуктивність мережі практично перестає впливати на час обробки транзакцій, і своєрідне розпилювання монолітних програмних блоків на мобільні OLE-сегменти приводить до значного підвищення ефективності роботи застосування в цілому. Більше того, трафік в мережі істотно зменшується і стає більше передбачуваним.

Найприйнятніший, і найбільш простий, ефективний підхід на думку більшості творців територіально-розподілених застосувань – це створення трирівневої архітектури. У цій моделі є користувач-клієнт і сервер бази даних, а також середній рівень, який діє як "діловий" або інтелектуальний сервер застосувань. Можна потім спростити програму користувача, переміщаючи ділову логіку системи на сервер застосувань, який міг би знаходитися на тому ж самому комп'ютері, що і сервер бази даних (чи на іншому комп'ютері). Часто це дає ефект прискорення розробки і пониження дистрибутивних витрат для обслуговування "товстих" клієнтів. Сервер застосувань здійснює запити до сервера бази даних за вимогами клієнтів і реалізує бізнес-логіку, що не лише спрощує розробку і супровід клієнтів, але також означає, що слід модифікувати тільки одну програму при зміні ділової логіки реалізації бізнес-процесів.

Щоб краще зрозуміти можливості розподіленої архітектури, розглянемо приклад з реального життя. Припустимо, є велика транспортна компанія M-Trans, яка забезпечує своїх клієнтів спеціальною програмою для відстежування шляху проходження вантажу. Коли ви відправляєте вантаж з Москви в Делі, вам дається спеціальний транспортний номер. Потім, якщо ви захочете дізнатися, що сталося з вашим вантажем на шляху слідування, ви просто завантажуєте програму (назвемо її MTrack) і вводите свій транспортний номер.

Діалоговий модуль зв'язується з центральною універсальною ЕОМ компанії і відшукує інформацію відносно поточного місця розташування і стану вашого вантажу. Наприклад, ви могли б дізнатися, що вантаж знаходиться нині у вузловому аеропорту відвантаження або що він був вже отриманий замовником. Електронний підпис замовника, природно, повинен свідчити про цей відрадний факт.

Поки усе звучить, подібно до банальної історії про типове обслуговування клієнтів на сервері. Проте проблема полягає в тому, що програма сильно залежить від типу присвоєного транспортного номера. Різні типи чисел означають різні типи послуг, що надавались користувачеві (наприклад, можливість упевнитися в підписі замовника товару, достовірності упаковки і т. д.). Щоб зробити MTrack настільки дружньою наскільки це можливо, розробники порахували, що користувач не повинен потребувати інтерпретації типу номера, який він отримав. Було б логічне покласти цю процедуру на центральну ЕОМ, але відносно неї у керівництва фірми існувала спеціальна політика і, крім того, вона була занадто завантажена іншими завданнями. Нічого не залишалося, як повернутися до ідеї відносно виконання цієї простої логіки на стороні користувача. Втім, проти такого рішення заперечувала група супроводу програмного забезпечення, мотивуючи тим, що, оскільки, можливо, розроблятимуться нові типи послуг, те програмне забезпечення не розпізнаватиме ці нові послуги і повинне час від часу модифікуватися. Оскільки програмне забезпечення в цій транспортній фірмі використовується більш ніж 50000 замовниками, це було б справжнім кошмаром. Тому було покінчено з ідеєю інтерпретувати транспортний номер користувача за допомогою відповідного діалогового вікна програми.

Витонченіше рішення в даному випадку могло б бути в створенні сервера застосувань на базі Windows в центрі даних компанії. Цей сервер застосувань міг би реалізувати усю ділову логіку, щоб розпізнавати різні типи транспортних номерів. Програма клієнта просто б зв'язувалася з сервером застосувань, а він вже інтерпретував би номер клієнта і здійснював відповідний запит на центральну ЕОМ. Це дійсно усунуло б проблему модифікації, тому що тільки сервер застосувань повинен був би модифікуватися, якби типи пропонованих послуг змінилися.

Цей приклад хороший тим, що показує, як потрібно використати сервер застосувань. Кращим варіантом завжди буде той, коли на сервері виконується програмний модуль, який може часто модифікуватися і який повинні викликати у своїй роботі одночасно тисячі клієнтів. Викликається він дистанційно і виконує корисну роботу. Клієнт і сам би міг мати на своїй машині цей модуль і усе для нього, клієнта, було б набагато простіше. Але річ у тому, що клієнтів – 50000, а модуль часто модифікується.

Усі інші аргументи на користь триланкової архітектури зводяться доки до того, що вона краща, тому що вона краща, – мовляв на клієнтові нічого не потрібно інсталювати, окрім інтерфейсів до віддаленого програмного сервера. Але при цьому, проте, не видно ніякої особливої вигоди, якщо програмне забезпечення, яке переноситься на сервер застосувань, рідко модифікується.

Завантаження мережі залишається тим же, а на сервер падає левова частка проблем, пов'язаних з масовим обслуговуванням клієнтів, управлінням транзакціями і відстежуванням черг.

Що мається на увазі під технологіями ActiveX

ActiveX служить одній єдиній меті: забезпечувати функціонування програмних компонентів усередині складених програмних контейнерів. Ці контейнери включають Web-браузери і інші засоби перегляду документів. ActiveX – технологія Microsoft, призначена для написання мережевих застосувань. Оскільки самим напрямом, що динамічно розвивається, в комп'ютерній індустрії є Інтернет, саме тут найприродніше можуть знайти своє місце програми, написані з використанням технології ActiveX. Не випадково останнім часом поняття ActiveX і Інтернету часто зустрічаються поруч. В той же час технологія ActiveX має значно більше універсальну сферу застосування.

Стандарт ActiveX дозволяє програмним компонентам взаємодіяти один з одним по мережі незалежно від мови програмування, на якій вони написані. За допомогою ActiveX можна "оживити" Web-сторінки ефектами мультимедіа, інтерактивними об'єктами або складними застосуваннями. ActiveX забезпечує деякі зв'язні засоби, за допомогою яких окремі програмні компоненти на різних комп'ютерах "склеюються" в єдину розподілену систему.

ActiveX включає в себе клієнтску і серверну частини, а також бібліотеки для разробника.

Керуючі елементи ActiveX (ActiveX Controls)

Що ж це таке насправді? Найбільш проста відповідь – ця нова назва для керуючих елементів OCX, або навіть для ще раніше них існуючих VBX. Будь-який готовий або створений вами управляючий елемент OLE – це вже ActiveX-елемент, який може використовуватися в програмах. Проте подібні OLE-елементи в усьому їх незліченному різноманітті навряд чи влаштують розробників для Web.

Типовий недолік існуючих OLE-елементів – їх значні розміри. Це обумовлено складністю структури OLE-інтерфейсів, а також тим фактом, що при підготовці в Microsoft бібліотек, використовуваних для генерації управляючих елементів, розміри їх не оптимізувалися. Якщо в системі користувача якийсь з цих елементів відсутній, доводиться завантажувати його через інтернет, отже, розмір управляючих елементів, Web-сторінок має бути якомога менше. З технічної точки зору ActiveX-елемент – це деякий COM-об'єкт, через основний OLE-інтерфейс якого, IUnknown, організовується доступ до інших інтерфейсів цього об'єкту.

По суті за допомогою ActiveX-елементів програміст створює високорівневий, придатний для багатократного використання об'єкт з деякою корисною функцією. Потім цей елемент може бути переданий (чи проданий) іншому програмістові, якому згодиться як деякий "будівельний блок". Більшість програмних інструментальних систем, таких як Delphi, Visual Basic, Visual C++ підтримують засоби взаємодії з ActiveX-елементами. В ролі ActiveX-елементів може бути усе що завгодно – від звичайної кнопки до повнофункціональної електронної таблиці. У продажі є тисячі таких елементів

від різних постачальників. Їх богаті функціональні можливості і різноманіття – окреме, найбільш важливе достоїнство платформи ActiveX.

Можна запитати, а яке відношення це має до Інтернету і баз даних? Відповідь і інтерпретації Microsoft звучатиме так: найпряміше і безпосереднє. За своєю суттю платформа ActiveX – це адаптація існуючих технологій Microsoft стосовно Web. По заявах Microsoft керуючі елементи ActiveX працюють швидше, ніж Java-аплети. Складні функції можна додавати клацанням миші. Головне, проте, не в тому, що ActiveX, як і Java-аплети, здатні оживити Web-сторінки. Куди важливіше той факт, що управляючі елементи ActiveX, дозволяють відвідувачам Web-вузла виконувати складні операції, отримувати потрібну інформацію від баз даних і від застосувань, працюючих на інших серверах або навіть на других Web-вузлах.

Оскільки ActiveX-файл є родичем VBX, він може використовуватися і в звичайному Visual Basic, і в мові сценаріїв VBScript. У таблиці 16.1 перераховані основні бібліотечні типи, що використовуються у Windows.

Таблиця 16.1.

Бібліотечні типи, використовувані в Windows

Управляючий елемент	**Розширення**	**Функція**
Бібліотеки (Dynamic link library), що динамічно підключається	.dll	Дозволяє користувачам діставати доступ до функцій, підпрограм і ресурсів з інших застосувань
Visual Basic	.vbx	Забезпечує той же призначений для користувача сервіс, що і DLL. Може застосовуватися в інтегрованих середовищах розробки, таких як Visual Basic 3.0, MSVC++ 1.52 і Delphi 1.0
ActiveX	.OCX	Забезпечує призначений для користувача сервіс, такий же, як DLL, і той же, що і VBX. На додаток, OCX є більш функціональними і краще використовує усі доступні йому ресурси. Підтримується практично усіма відомими RAD-системами

Охарактеризуємо коротко базові поняття і ключові об'єкти, з яких будуються керуючі елементи ActiveX.

Контейнер і взаємодія з ним

Управляючий елемент ActiveX не може існувати сам по собі, він завжди "живе" усередині свого контейнера і тісно взаємодіє з ним. Через властивості оточення (ambient properties) об'єкт має доступ до інформації про поточний стан свого "власника". Контейнер підтримує додаткові методи, властивості і події,

які для розробника виглядають як частину інтерфейсу елементів управління ActiveX.

Властивості оточення

Інформація про стан контейнера доступна ActiveX-елементу через об'єкт AmbientProperties, посилання на який може бути отримане через властивість Ambient об'єкту UserControl.

Наприклад, властивість UserControl.Ambient.BackCoior відображає поточне значення кольору фону контейнера і зазвичай використовується при відображенні ActiveX-елемента. Грамотно написаний ActiveX-елемент повинен поводитися відповідно до поточного стану контейнера. Повідомлення userControl_Ambientchanged сповіщає управляючий елемент ActiveX про зміну значення якої-небудь властивості оточення контейнера.

Додаткові властивості

Додаткові властивості (extender properties) надаються контейнером, але зовні виглядають як частину інтерфейсу елементів управління ActiveX. Розробник ActiveX-елемента має доступ до додаткових властивостей через властивість Extender об'єкту userControl. Специфікація елементів управління ActiveX вимагає, щоб усі контейнери підтримували Наступні Додаткові властивості: Name, Visible, Parent, Cancel Default. Для доступу до додаткових властивостей завжди використовується механізм пізнього зв'язування (late - bound), оскільки на момент компіляції невідомо, з яким контейнером належить працювати ActiveX-елементу.

Коли користувач звертається до властивості (методу) ActiveX control, то першим управління отримує об'єкт Extender. Якщо він не підтримує цю властивість (метод), то викликається обробник ActiveX Controls.

Об’єкт UserControl

ActiveX-елемент, створений на Visual Basic, завжди містить (агрегує) об'єкт userControl. Розміщення і налаштування властивостей складових (constituent) ActiveX-елемента проводиться за допомогою редактора властивостей.

Об'єкт userControl має стандартний інтерфейс. Ви можете написати свої обробники для подій, генерованих цим об'єктом, надати для використання або перевизначити стандартні властивості. Так само ваш ActiveX-елемент може експортувати методи, властивості і події складових компонентів.

Ключові події

В процесі свого існування ActiveX-елемент отримує сповіщення про ряд подій, генерованих об'єктом UserControl. Найбільш важливими є:

– **initialize.** Найперша подія, завжди відбувається при створенні ActiveX-елемента. У цей момент об'єкт ще не має зв'язку зі своїм контейнером.

– **initProperties.** Ця подія сповіщає про розміщення нового ActiveX-елемента у формі. У момент його приходу вбудований об'єкт вже має зв'язок зі своїм контейнером. Як правило, обробник події виконує початкову ініціалізацію властивостей ActiveX-елемента.

– **ReadProperties.** При завантаженні форми (як у **DesignMode,** так і в **RunMode)** вбудовані в неї елементи створюються наново і їх властивості

ініціалізувалися значеннями, збереженими у файлі з розширенням .frm. Подія ReadProperties сповіщає ActiveX-елемент про необхідність виконати ініціалізацію. У цей момент об'єкт має зв'язок зі своїм контейнером.

CORBA и Enterprise JavaBeans – лідери найновіших технологій. Навіщо потрібна CORBA

Відразу обмовимося, що під CORBA ми розумітимемо CORBA/IIOP, тому що сама специфікація CORBA не стандартизує мережевий протокол передачі даних між клієнтом і сервером. Кожен виробник брокера об'єктних запитів, який, по суті і є CORBA, може запропонувати власний протокол транспортної служби передачі даних. Проте для забезпечення інтероперабельності брокерів запитів від різних виробників OMG (Object Management Group – міжнародна некомерційна організація, в яку входять промислові компанії і компанії, що займаються створенням програмного забезпечення) розробив загальний протокол GIOP. Його відображення в TCP/IP дістало назву протоколу Internet Inter ORB Protocol (IIOP). Завдяки цьому, будь-яке застосування CORBA/IIOP повинне працювати і взаємодіяти з іншими CORBA/IIOP-додатками незалежно від платформ, виробників програмного забезпечення і операційних систем.

CORBA (Common Object Request Broker Architecture) – це технологія для світу розподілених інформаційних об'єктів, що тісно взаємодіють між собою у рамках програми, що управляє ними, яка віртуально з цих об'єктів і складається. Так от, CORBA – це архітектура, яка з максимальними зручностями забезпечує створення і функціонування програм, званих CORBA - додатками. Останнім часом багато RAD-системи: Delphi, включають у свій арсенал підтримку CORBA.

Втім, традиційні RAD-системи, подібні Visual Basic, не потребують CORBA, тому що орієнтуються на наявний в Microsoft власний брокер об'єктних запитів, існуючий під іменем DCOM. Надалі виробникам програмних продуктів для програмістів стане все важче і важче підтримувати обидві технології. Delphi-програмісти можуть заспокоювати себе тим, що є можливість працювати і з CORBA, і з DCOM.

Звичайно ж, CORBA – більше масштабований, відкритіший і грандіозніший проект, ніж DCOM. CORBA розглядає вже існуючі застосування, як деякі метафори, які легко можна адаптувати і використати, правильно і одного разу описавши їх інтерфейси за допомогою спеціальної мови описів IDL (Interface Definition Language – мова опису запитів), для якого існують компілятори у більшість сучасних мов програмування. CORBA є прообразом нашого об'єктного майбутнього. У комп'ютерному світі абсолютно нормальним вважається факт, що програмісти усіх країн по мільйону разів переписують наново один і той же алгоритм, якщо він вимагає внесення деяких невеликих змін. Навіщо?

Адже у будь-якому випадку, безумовно, існує кращий варіант того фрагмента програмного коду, який переписується із самого початку людьми, ніби вони не знають про це. Чому, в порівнянні з програмістами, так далеко уперед пішли інженери комп'ютерних систем? Та тому, що вони вже дуже

давно використовують об'єктну технологію у вигляді незмінних конструкторських елементів ("чіпів"). А програмісти по-старому розпочинають з пісочку: спочатку просіємо пісочок, потім додамо в нього глину, після цього виготовимо цеглу, а вже потім з тієї цегли побудуємо конструкцію. Кустарне виробництво.

Що означають об'єктні технології в програмуванні, і не лише в програмуванні, а ширше, – в усій комп'ютерній творчості в епоху безроздільного панування Мережі? Ми вимушені визнати, що сучасне виробництво начисто позбавлене індивідуалізму і визнає тільки те, що може бути розмножене. Найкращим варіантом вважається той, який ви не створюєте, а лише настроюєте, компонуючи його з різного набору об'єктів, описуючи їх властивості і задаючи різні методи, що виконують той або інший вид роботи. Виграш продуктивності виходить колосальний. У всьому панує єдиний принцип – повторне використання компонентів. Ваша праця, якщо ви є творцем нових інформаційних об'єктів і одночасно клієнтом системи об'єктної реалізації, легко може перетворитися на об'єкт: процес або програму, які можуть використати інші клієнти і об'єкти з метою використання ваших методів, якщо вони містять в собі щось нове, незалежно від ваших бажань, місцезнаходження і часу.

Звичайно, було б ідеально, якби повторно використовувані об'єкти самі б не були об'єктами, а їх творець мав права на плоди своєї творчості. Але ж неможливо уявити собі, що б творець якого-небудь об'єкту творив у вакуумі, він усього лише наслідує структуру якого-небудь опису, якого-небудь шаблону, класу або групи класів – пакетів. Те що зараз відбувається з програмними модулями, що оформляються на універсальній мові програмування в JavaBeans або в ActiveX, те ж, по суті, відбувається і з авторськими текстами, що перетворюються не на індивідуальні плоди творчості, а в один нескінченний і універсальний коментар. Який сенс наново описувати те, що вже одного разу описано? Проте це робиться, і текст, проходячи через безліч об'єктів, що інтерпретують його, постійно шліфується, до тих пір, поки не набуває абсолютної форми. Якщо раніше Мережа ще носила на собі яскравий відбиток індивідуалізму її творців, то тепер вона все більш і більш перетворюється на деякий Універсум, який за визначенням не терпить нічого, що виходить за його межі. Усе або майже усе в Мережі придбаває статус анонімності. Використовуються вже не повноцінні імена, а усього лише ідентифікатори, не індивідуальні паролі, а сертифікати. Творців не може бути надто багато. Врешті-решт єдиним творцем усього стає сама Мережа.

У цих умовах розмови про авторські права стають просто смішні. Ніхто нічого не винаходить, він усього лише наслідує. Поняття поліморфізму також є невід'ємною рисою систем об'єктних реалізацій. Ви просто оголошуєте себе спадкоємцем яких-небудь загальновживаних класів: понять, конструкцій, образів, текстів і модифікуєте їх відповідно до своїх установок. Творець набуває себе усього лише у витяганні нових сенсів із старих конструкцій – форм і інструментів у нього цілком вистачає. У успадковному класі при цьому нічого не змінюється, проте новий клас, будучи спадкоємцем старого, по волі

свого творця, містить в собі перевизначені ключові системні і смислові блоки, так що вони стають поліморфними, т. е. поводяться і представляються абсолютно по-різному, залежно від того, хто і яким чином їх затребував.

Приклади абсолютно прості: ви включаєте свій телевізор таким самим способом, як праску, електропіч або настільну лампу. Усі предмети, що відносяться до класу "Прилад", містять метод "включити", але в процесі включення, залежно від типу приладу (підкласу), єдиний метод "включити" поводиться поліморфно, тобто викликає абсолютно різні електричні і блокові функції: для праски – одні, для телевізора – інші. Можна так перевизначити класи і методи в телевізорі новітньої конструкції, що він поводитиметься абсолютно по-різному, залежно від того, ХТО його включив, тобто телевізор поведеться поліморфно. Більше того, існують програмні засоби (Java+XML), які здатні динамічно модифікувати контент. Попри те, що Java і XML самі по собі досить складні, інструменти, засновані на них, для звичайного користувача надзвичайно прості і приємні, бо вони відводять його від необхідності знати внутрішні механізми інформаційного джерела і призводять до дистанційного пульта управління ними.

Точно так, як і він зараз управляє своїм телевізором, не замислюючись про його технічний склад, в майбутньому, завдяки новим, усе більш досконалим технологіям, він буде здатний управляти інформацією, структура якої визначається побажаннями клієнта. З точки зору систем об'єктної реалізації, люди також представляють з себе об'єкти з програмованими поведінковими властивостями. Вони самі, будучи підключені до електронних систем, неодноразово дефініюють (перевизначають) свої інформаційні запити до ресурсів, які система запам'ятовує і згодом використовує. Завжди, як тільки людина вступає у взаємодію з програмованими об'єктами, він сам стає об'єктом, він сам стає програмованим.

Для постмодерністської епохи, завершенням якої є Інтернет, цінності в традиційному розумінні не є цінностями, бо вона маніпулює ними і будує з них структури більш високого порядку. Плоть і кров пішли з цивілізованих людей, залишилася лише електрика. Безглуздо заявляти свої права на конструкції, які існують як візерунки в калейдоскопі. Усе стає віртуальним, усе стає відносним, а точка відліку – Мережа. Сучасні люди вже не здатні круто змінити парадигму свого існування. Творчість в епоху Інтернету є витяганням усе нових і нових сенсів з відомих речей і постійне оновлення простору контенту.

Сила CORBA в тому, що це система, що самоописується і самодокументується. Вона була розроблена для того, щоб дозволити створювати інтелектуальні компоненти, які можуть запросити інформацію один про одного і потім її ефективно використати. Число взаємодіючих об'єктів при цьому не обмежене, способи існування і місцезнаходження об'єктів жорстко не визначені. Будь-який об'єкт може бути затребуваний іншим об'єктом або програмою в довільній точці Мережі, і це дуже сильно нагадує телепортацію.

Архітектура CORBA.

Чотири головні елементи CORBA архітектури показані на рис. 16.4.

Рисунок 16.4 – Головні елементи CORBA

– **Брокер об'єктних запитів (ORB – Object Request Broker)** визначає механізми взаємодії об'єктів в різнорідній мережі. Посередник об'єктних запитів, **ORB,** або просто брокер, – це логічне ядро системи. Саме він дозволяє об'єктам посилати запити і отримувати відповіді від інших об'єктів в мережі. ORB – це проміжне програмне забезпечення, яке встановлює відношення між об'єктами в розподіленому середовищі. Брокер по посиланню знаходить на сервер, що містить об'єкт-адресат (object implementation – реалізація об'єкту); активізує його, доставляє до нього запит; передає параметри і викликає відповідний метод, після чого повертає результат клієнтові. Ролі клієнта і сервера при цьому можуть мінятися. Взаємодія може бути встановлена між безліччю об'єктів. OMG розробив спеціальну мову верхнього рівня для опису ORB і інших компонентів CORBA – так званий IDL (Interface Definition Language – мова опису інтерфейсів). У ній немає присвоєнь, звичайних мовних конструкцій, типу if чи while, функцій і логічних переходів і т. д. IDL – це мова декларацій, вона описує батьківські класи, події і методи. Кожен інтерфейс визначає операції, які можуть бути викликані клієнтами, але не те – яким чином ці операції виконуються зовні IDL і CORBA. У цьому і привабливість – можна змусити працювати старі програми, якщо правильно описати їх інтерфейси і використати компілятор з IDL, який має відображення в конкретну мову програмування. Важливо зрозуміти, що є два типи інтерфейсів: динамічні і статичні. Перші визначаються на стадії розробки застосування і компілюються разом з ним, другі конструюються у момент виконання, на працюючому застосуванні.

– **Сервіси об'єктів (Object Services)** є основними системними сервісами, які розробники використовують для створення застосувань.

– **Універсальні засоби (Common Facilities)** є також системними сервісами, але більш високого рівня, орієнтованими на підтримку призначених для користувача застосувань, таких як електронна пошта, засоби друку і т. д.

– **Прикладні об'єкти (Application Objects)** використовуються в конкретних прикладних завданнях.

Так, все-таки, навіщо потрібна CORBA?

Розподілені застосування

CORBA-архітектура забезпечує каркас для розробки і виконання розподілених застосувань. Але виникає нове питання – а навіщо взагалі

потрібний цей розподіл? Як тільки ви зіткнетеся з ним, то побачите перед собою величезну купу нових проблем. Проте іноді немає іншого вибору: деякі застосування просто вимагають бути розподіленими з наступних причин:

– дані, використовувані застосуванням, – розподілені;
– обчислення – розподілені;
– користувачі застосування – розподілені.

Розподілені дані

Деякі застосування повинні виконуватися на безлічі комп'ютерів, тому що дані, до яких вони повинні звернутися, існують також на безлічі комп'ютерів. Власник може дозволити дистанційний доступ до даних, але зберігаються вони як локальні. Такі історичні (чи інші) причини, що визначають розподілену організацію даних.

Розподілені обчислення

Деякі застосування виконуються на безлічі комп'ютерів, щоб скористатися перевагою паралельного обчислення для вирішення специфічних завдань, наприклад, пов'язаних з дешифруванням. Інші застосування також можуть виконуватися на безлічі комп'ютерів, але щоб використати переваги деяких специфічних систем для реалізації різних частин свого алгоритму. Розподілені застосування завжди виграють при використанні необмеженої масштабованості і неоднорідності розподіленої системи.

Розподілені користувачі

Деякі застосування розподілені по безлічі комп'ютерів, тому що користувачі зв'язуються і взаємодіють один з одним через це застосування. Кожен користувач виконує фрагмент розподіленого застосування на його власному або чужому комп'ютері і активно використовує загальнодоступні об'єкти, які зазвичай виконуються на одному або більшій кількості серверів. Типова архітектура для цього виду застосувань ілюструється на рис. 16.5.

Як вже говорилося, CORBA є ідеальною архітектурою для створення таких застосувань. Актуальність її в наші дні активно затребувана розвитком Інтернету і систем електронної комерції, де багато функцій має бути розподілені по мережі і де є удосталь безліч вже працюючих об'єктних модулів, існуючих у вигляді ActiveX, або JavaBean-компонентів.

Брокер об'єктних запитів, ORB, ставить запит об'єкту і повертає будь-який результат клієнтові (рис. 16.6).

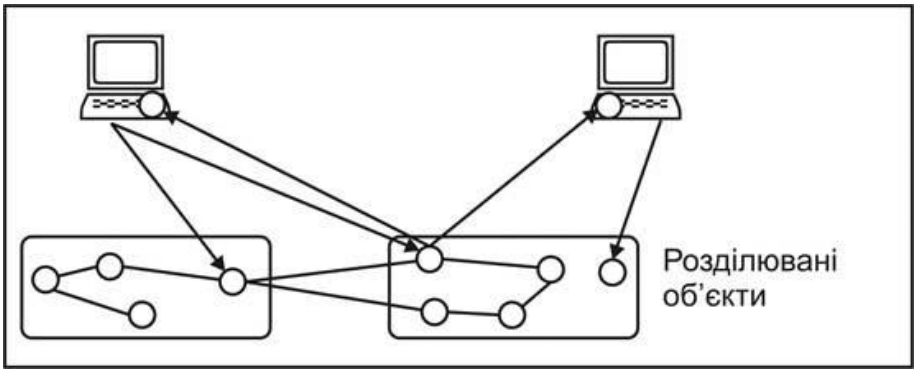


Рисунок 16.5. – Розподілені користувачі

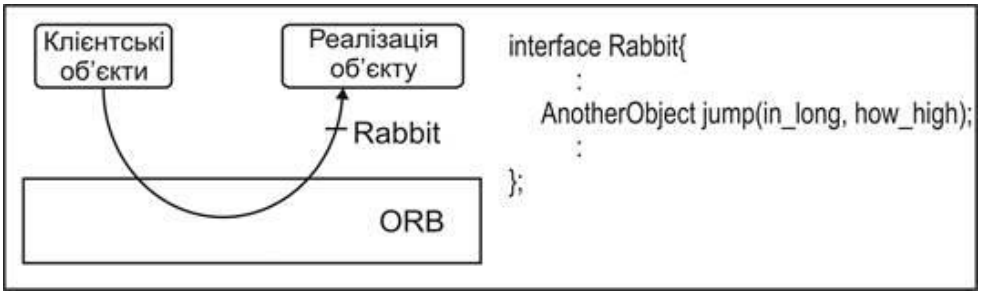


Рисунок 16.6. – Механізм взаємодії: клієнт запитує об'єкт через посередника ORB

Від клієнта до сервера

Оскільки CORBA є стандартом для створення розподілених застосувань, коли комп'ютери через видалений доступ викликають програмні методи і об'єкти один одного, в ній чітко прописаний рівень міжмережевих взаємодій поза всякою залежністю від місцезнаходження, мови і архітектури.

Можна чітко сформулювати ключові особливості CORBA-стандарту:

– CORBA-об’єкти можуть знаходитись в будь-якому месці мережі;

– CORBA-об’єкти можуть взаємодіяти з іншими CORBA -объектами на будь-яких платформах;

– CORBA-об’єкти можуть бути написані на будь-якій мові програмування, для якого є інтерфейс, що реалізовується IDL-компілятором (такі є для Java, C++, C, SmallTalk, COBOL і Ada).

Використовуючи CORBA,можна створювати розподілені системи, де різні компоненти (інтерфейси користувачів, бізнес-алгоритми, доступ до баз даних і т. д.) упаковуватимуться в окремих програмах, що виконуються на різних машинах. Кожен компонент зв'язуватиметься з іншим тільки через оголошений інтерфейс. При цьому можна відлагоджувати і підтримувати частини програм окремо.

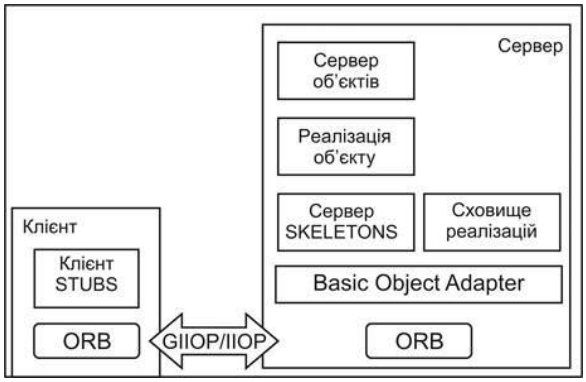


Рисунок 16.7. – Архітектура CORBA-сумісного застосування, що викликає безліч об'єктів з безлічі комп'ютерів

Клієнтська частина CORBA

Коли CORBA-сумістний об'єкт використовує метод, що знаходиться в іншому об'єкті, він викликає стаб (stub), тобто деяку порожню оболонку необхідного об'єкту, його інтерфейсну частину, що знаходиться в спеціальному stub-файлі. Цей стаб використовує локальний ORB, посилає запит і отримує відповідь. Об'єкту немає ніякої необхідності знати подробиці відносно ORB або

дійсного місцезнаходження сервера. Усі CORBA-клієнти і сервери з'єднуються через брокерів ORB, використовуючи їх як посередників.

JDK містить ORB, який дозволяє створювати CORBA-сумісні програми на Java.

Клієнтський стаб створюється при першому визначенні серверного інтерфейсу в клієнтові за допомогою мови визначення інтерфейсів IDL. CORBA IDL визначає тип об'єкту, його методи, властивості і т. д. CORBA IDL – це абсолютно нейтральна і чисто декларативна мова.

Стаб потім використовує локальний ORB, посилає запит і отримує відповідь. ORB надає своєрідну шину для проходження обміну структурованими даними. Ця шина використовує IIOP як транспортний протокол між брокерами ORB.

Серверна частина CORBA

– При витяганні запиту IIOP сервер ORB використовує так званий Basic Object Adapter (BOA) в комбінації з депозитарієм виконання (Implementation Repository – набором конкретних класів і бібліотек, наявних на сервері) для передачі параметрів запиту до необхідного серверного об'єкту через серверний стаб. Іноді запитують – навіщо так складно? Відповідь проста, щоб легше було програмувати незалежні частини коду, зосереджуючись тільки на суті коду, а не на вирішенні системних питань, пов'язаних з мережевою специфікою і міжпрограмній взаємодії. Про них користувач взагалі не повинен замислюватися, в точності так само, як він не замислюється про внутрішній устрій телефонної трубки і телефонної станції, через які проходить голосовий сигнал.

– **Basic Object Adapter BOA –** це набір системних сервісів ORB для встановлення з'єднання з серверними об'єктами по їх ідентифікаторах (об'єктні посилання).

– **Server IDL Stub** Служить для забезпечення інтерфейсу до кожного сервісу. На серверній стороні такої стаб, щоб відрізнити його від клієнтського стаба, називається скелетон.

– **Implementation Repository** Це набір класів на стороні сервера, доступний усім розподіленим клієнтам через систему об'єктних посилань.

Сервіси CORBA

Важливою частиною стандарту CORBA є визначення набору розподілених послуг (сервіси CORBA), щоб підтримувати інтеграцію і взаємодію розподілених об'єктів. Вони визначені як стандартні об'єкти CORBA з інтерфейсами IDL і іноді ще згадуються як об'єктні сервіси (рис. 16.8).

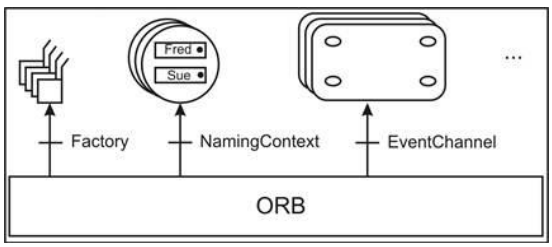


Рисунок 16.8. – Об'єктні сервіси

Є безліч CORBA-сервісів. Найпопулярніші з них наведені в таблиці 16.2.

Таблиця 16.2.

Найбільш популярні сервіси CORBA

Сервіс	Опис
Життєвий цикл об’єкта (Object life cycle)	Визначає, яким чином CORBA-об’єкти будуть створені, видалені, переміщені або скопійовані
Іменування (Naming)	Визначає спосіб символічного позначення CORBA – объектов
Події (Events)	Обробка подій
Відношення (Relationships)	Визначають стосунки між об'єктами (головний, підлеглий, зв'язковий і т. д.)
Перетворення об'єктів з однієї форми в іншу (Externalization)	Координує перетворення об'єктів із зовнішніх форм у внутрішні і навпаки
Транзакції (Transactions)	Координують доступ до CORBA-об’єктів
Контроль доступу (Concurrency Control)	Забезпечує обслуговування блокування об'єктів CORBA
Властивість (Property)	Підтримує асоціацію пари: ім'я-значення для об'єктів CORBA
Продавець (Trader)	Підтримує пошук CORBA-об’єктів, заснований на афішованих властивостях об'єкту
Запрос (Query)	Підтримка запитів до об'єктів

Таблиця 16.3.

Найбільш відомі реалізації CORBA

ORB	Опис
Java ORB	Java 2 ORB поставляється у складі Sun's Java SDK. Там відсутні деякі особливості повної специфікації
VisiBroker for Java	Популярний Java ORB з Inprise Corporation. VisiBroker вбудований у багато інших продуктів. Приміром, саме він був вбудований у браузер Netscape Communicator.
OrbixWeb	Добротний Java ORB з lona Technologies (для Unix)
WebSphere	Потужний сервер застосувань зі вбудованим ORB від IBM
Free or shareware ORBs	CORBA-реалізації для різних мов програмування доступні для завантаження по мережі Інтернет з різних джерел

CORBA-продукти

Не забувайте, що CORBA – це усього лише специфікація. Що стосується програмних виробів, що реалізовують архітектуру CORBA, то їх є множина. Окремі продавці забезпечують свої брокери CORBA і IDL для відображення в різні мови програмування. Усі вони зобов'язані підтримувати Java. У таблиці. 16.3 приведені найбільш відомі реалізації CORBA.

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. I. Foster, C. Kesselman. The Grid: Blueprint for a New Computing Infrastructure. – San Francisco, Morgan Kaufmann Publishers, 1998. – pp. 259 – 278.
2. D. Clark. Face-to-Face with Peer-to-Peer Networking. // Computer, Vol. 34, No. 1, 2001. – pp. 18 – 21.
3. I. Foster, C. Kesselman, S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. // International J. Supercomputer Applications. – 2001, 15 (3), pp. 12 – 15.
4. J. Treadwell, M. Behrens, D. Berry et al. Open Grid Services Architecture Glossary of Terms. – Global Grid Forum, 2005. – p.4 – 8.
5. http://www1.cnri.reston.va.us /gigafr/- CNRI, Corporation for National Research Initiatives, Gigabit Testbed Initiative Final Report,December, 1996.
6. I. Foster. Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment / I. Foster, J. Geisler, W. Nickless, W. Smith, S. Tuecke. // John Wiley and Sons, 2003. – pp 99 – 111.
7. http://www.geant.net – офіційний сайт проекту GEANT.
8. http://lcg.web.cern.ch/LCG - офіційний сайт проекту LCG.
9. I. Foster. The Nexus Approach to Integrating Multithreading and Communication / I. Foster, C. Kesselman, S. Tuecke. – // J. Parallel and Distributed Computing, 37 : 70 – 82, 1996. – 105 p.
10. http://www.eurogrid.org – офіційний сайт Європейського проекту EUROGRID.
11. http://www.ebi.ac.uk – сайт Європейського Інституту Біоінформатики.
12. http://www.ogf.org/ – сайт спільноти Open Grid Forum.
13. I. Foster. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration / I. Foster, C. Kesselman, S. Tuecke, J.M. Nick. – San Francisco, Morgan Kaufmann Publishers, 2002. – pp. 54 – 62.
14. http://www.eu-egee.org/ – офіційний портал проекту Enabling Grids for E-sciencE (EGEE).
15. http://www.globus.org – офіційний сайт проекту Globus.
16. http://www.griphyn.org – сайт проекту GriPhyN.
17. http://www.ppdg.net – сайт проекту PPDataGrid.
18. http://www.astrogrid.ac.uk – сайт проекту AstroGrid.
19. http://www.combechem.org – сайт проекту Comb-e-Chem.
20. http://montage.ipac.caltech.edu/ – сайт проекту Montage, An Astronomical Image Mosaic Service for the National Virtual Observatory.
21. www.epsrc.ac.uk – сайт Британської Інженерно-фізичної Науково-дослідної Спілки.
22. http://gridport.npaci.edu/gamess – сайт проекту GAMESS.
23. http://www.ibm.com – офіційний сайт компанії IBM.
24. http://www.bytemag.ru/ – сайт російського відділення журналу BYTE.
25. http://www.eumedgrid.org/ – офіційний портал проекту EUMed-Grid.
26. http://www.gridcomputing.com/ – сайт Grid Computing Info Centre.
27. http://www.ivdgl.org – сайт Міжнародної Віртуальної Лабораторії Grid Даних.

28. http://eu-datagrid.web.cern.ch – сайт Європейського проекту DataGrid.
29. http://root.cern.ch/ – сайт системи ROOT.
30. http://www.d-grid.de – офіційний сайт проекту Die Deutsche Grid-Initiative.
31. http://www.geodise.org/ – портал системи GEODISE.
32. http://grid.ntu-kpi.kiev.ua/ – сайт проекту „Створення Національної GRID інфраструктури України для наукових досліджень”, Інститут прикладного системного аналізу НТУУ "КПІ".
33. http://lhcgrid.web.cern.ch/LHCgrid – сайт проекту LHC Computing.
34. http://www.teragrid.org/index.php – офіційний сайт проекту TeraGRID.
35. http://www.beingrid.eu/ – офіційний сайт проекту BEinGRID (Business Experiments in GRID).
36. IBM and SAS Working Together to Promote Business Innovations with Grid Cimputing. SAS White Papers, 2006.
37. http://www.garr.it/conf07/slide/pelfer.pdf – сайт проекту EUMEDGrid з проектом ArchaeoGRID.
38. http://vdt.cs.wisc.edu/. – T.Yamashita, Grid environment “CAD-Grid” for mobile communication system simulation. PSE Workshop in Sapporo, July 2004. – pp / 31 – 36.
39. http://vdt.cs.wisc.edu/ – Virtual Data Toolkit.
40. http://www.crossgrid.org/ – сайт проекту CrossGrid.
41. http://www.servogrid.org – сайт Solid Earth Research Virtual Observatory.
42. http://www.earthsystemgrid.org – сайт проекту DoE Earth Systems (Climate) Grid.
43. http://www.ncbiogrid.org – сайт Biomedical Informatics Research Network BIRN Grid.
44. http://www.gria.org – European Grid Resources for Industrial Applications.
45. http://www.escience-grid.org.uk/docs/briefing/nigridp.htm – List of Grid Projects.
46. http://www.research-councils.ac.uk/ escience/documents/gridteam.htm – UK e-Science Network.
47. http://gridcomputsng.org./grid 2003 – сайт з описом проекту grid 2003.
48. http://www.pdb.org – сайт Банку Даних Протеїнів (PDB).
49. http://www.ipg.nasa.gov/. – сайт NASA Information Power Grid.
50. http://www.sura.org – сайт проекту SURAgrid.
51. http://www.lagrid.fiu.edu – сайт проекту LA Grid.
52. http://www.mygrid.info/ – сайт MyGrid – Directly Supporting the e-Science.
53. http://pcalimonitor.cern.ch/map.jsp – сайт MonALISA Repository for ALICE.
54. http://www.climateprediction.com – Ensemble Climate Prediction.
55. http://www.us-vo.org/. – National Virtual (Astronomical) Observatory.
56. http://www.eso.org/avo/. – European Astrophysical Virtual Observatory.
57. http://www.mssl.ucl.ac.uk/grid/egso/egso top.html. – European Grid of Solar Observations EGSO.

58. http://www.neesgrid.org/ – NEES Grid, National Virtual Collaboratory for Earthquake Engineering Research.
59. http://www.servogrid.org. – Solid Earth Research Virtual Observatory.
60. http://www.earthsystemgrid.org/ – DoE Earth Systems (Climate) Grid.
61. http://www.crossgrid.org/ – European Cross Grid Infrastructure Project.
62. http://nws.cs.ucsb.edu/. – The Network Weather Service.
63. http://icl.cs.utk.edu/netsolve/ – NetSolve/GridSolve overview.
64. J. Dongarra, K. Seymour. GridSolve: The Evolution of A Network Enabled Solver. – University of Tennessee, 2006. – 25 p.
65. http://icl.cs.utk.edu/netsolvedev/documents/ug/html/UG.html – Users' Guide to NetSolve V2.0.
66. http://www.itrd.gov/pubs/blue02/national-grand.html – офіційний сайт проекту National Grand Challenge Applications.
67. http://www.cs.york.ac.uk/DAME – офіційний сайт проекту DAME.
68. http://www.npac.syr.edu/factoring.html – офіційний сайт проекту FAFNER.
69. https://gilda.ct.infn.it – офіційний сайт проекту GILDA.
70. www.globus.org/research/ papers/ogsa.pdf – Foster, I., Kesselman, C., Nick, J.M., & Tuecke, S. The Physiology of the Grid – An Open Grid Service Architecture for Distributed Systems Integration.
71. http://legion. virginia.edu/index.html – Legion Worldwide Virtual Computer Home Page.
72. http://eu-dataGrid.web.cern.ch/ – офіційний сайт проекту DataGrid.
73. Platform Grid Computing
74. http://www.cs.wisc.edu/condor/. – офіційний сайт проекту Condor.
75. http://entropia.com/ – офіційний сайт проекту Entropia.
76. http://www.jxta.org. – JXTA Peer-to-Peer Technology.
77. Петренко А.І. Інтелектуальна обробка інформації. // Системний аналіз і інформаційні технології. – Київ, № 4, 2008.
78. http://www.accessGrid.org – офіційний сайт проекту Access Grid.
79. Berners-Lee,T. The Semantic Web / Berners-Lee,T., Hendler,J. and Lassila. – // Scientific American, May, 2001. – pp.598 – 604.
80. http://www.ipg.nasa.gov - NASA Information Power Grid.
81. http://www.fipa.org/ – The Foundation for Physical Agents.
82. Housley, R.. RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List Profile / Housley, R., Polk, W., Ford, W., and Solo, D. – Standart. – 2002.
83. Next Generation Grids 2.Requirements and Options for European Grids. Research 2005 – 2010 and Beyond Expert Group Report, July 2004.