2. Чуприна О.С. Гібридні алгоритми розв’язання задач двовимірного пакування з урахуванням технологічних обмежень. *Системні технології*. 2019. № 2 (121). С. 45–53.

УДК 004.738.5:004.451:621.397.13

П.В. Свінціцький, Н.М. Пановик, О.В. Доберчак, І.О. Боднарчук
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

СУЧАСНА АРХІТЕКТУРА СТРІМІНГОВИХ ПЛАТФОРМ В РЕАЛЬНОМУ ЧАСІ

P.V. Svintsitsky, N.M. Panovyk, O.V. Doberchak, I.O. Bodnarchuk

MODERN ARCHITECTURE OF REAL-TIME STREAMING PLATFORMS

Потокове передавання даних у реальному часі передбачає безперервне введення, обробку та виведення даних, що відбувається майже миттєво. Воно дозволяє отримувати, аналізувати та реагувати на дані в міру їх створення, надаючи цінну аналітику без затримки. Ця технологія має вирішальне значення в умовах, коли своєчасна інформація є критично важливою, що дозволяє швидко приймати рішення та проводити аналітику в реальному часі.

Основною характеристикою систем потокового передавання даних у реальному часі є їхня здатність обробляти величезний обсяг даних з низькою затримкою [1, 3]. Це скорочує час між генерацією даних та отриманням корисної аналітики, що робить їх безцінними для різних галузей, таких як фінанси, охорона здоров'я та технології. Незалежно від того, чи йдеться про потокове передавання даних фондового ринку, чи про моніторинг пристроїв Інтернету речей, потокове передавання даних у реальному часі забезпечує ефективність та швидку реакцію на обробку даних.

Наведемо декілька загальних прикладів використання стрімінгу в реальному часі.

1. Моніторинг соціальних мереж.

Платформи соціальних мереж постійно генерують величезні обсяги даних. Потокове передавання даних у режимі реального часу дозволяє компаніям моніторити канали соціальних мереж на предмет згадок брендів, настроїв клієнтів та нових тенденцій. Це допомагає організаціям проактивно взаємодіяти з клієнтами, керувати своєю онлайн-репутацією та коригувати маркетингові стратегії на ходу.

2. Обробка фінансових даних.

У фінансових послугах потокове передавання даних у режимі реального часу використовується для аналізу ринку, торгових стратегій та управління ризиками. Фондові біржі та трейдери покладаються на дані в режимі реального часу, щоб приймати обґрунтовані рішення та здійснювати угоди. Ця можливість дозволяє фінансовим установам миттєво реагувати на коливання ринку, максимізуючи можливості отримання прибутку та мінімізуючи збитки.

3. Виявлення шахрайства.

Системи виявлення шахрайства використовують потокове передавання даних у режимі реального часу для виявлення підозрілої діяльності в міру її виникнення. Аналізуючи моделі транзакцій та поведінку користувачів, ці системи можуть виявляти аномалії, що свідчать про шахрайські дії. Такий моніторинг у режимі реального часу допомагає запобігти шахрайству, дозволяючи вживати негайних заходів, таких як блокування транзакцій або сповіщення користувачів.

4. Прогнозне обслуговування.

Прогностичне обслуговування використовує потокове передавання даних у режимі реального часу для моніторингу продуктивності обладнання та прогнозування збоїв до їх виникнення. Датчики на обладнанні безперервно надсилають дані про такі параметри, як температура, тиск і вібрація. Ці дані обробляються в режимі реального часу для виявлення аномалій та прогнозування потенційних поломок, що дозволяє своєчасно втручатися в технічне обслуговування.

Далі опишемо основні компоненти архітектури потокової передачі даних у режимі реального часу.

1. Джерело даних.

Компонент джерела даних – це місце, звідки дані походять. Це можуть бути різні системи, програми або пристрої, включаючи датчики, журнали, бази даних та платформи соціальних мереж. Ці джерела безперервно генерують необроблені дані, передаючи їх у конвеєр потокової передачі даних. Ефективне керування кількома джерелами даних має вирішальне значення для безперебійного потоку даних. Інтеграція різноманітних джерел даних часто вимагає конекторів або API для забезпечення сумісності та передачі даних.

2. Забір потоку.

Забір потоку – це процес захоплення та імпорту потоків даних на платформу потокової передачі даних. Для виконання цього завдання зазвичай використовуються такі технології, як Apache Kafka, Amazon Kinesis та Azure Event Hubs. Вони можуть забирати величезні обсяги даних у режимі реального часу, гарантуючи, що жодні дані не будуть втрачені під час передачі. Ефективний забір потоку вимагає низької затримки та високої пропускної здатності для підтримки цілісності потоку даних. Цей етап також включає завдання попередньої обробки (фільтрація, перетворення, тощо.

3. Потокове сховище.

Після отримання дані необхідно зберігати для подальшого аналізу. Рішення для потокового сховища, такі як внутрішнє сховище Apache Kafka, AWS S3 та Azure Blob Storage, підтримують тимчасове або довгострокове зберігання потоків даних. Ці системи обробляють великі обсяги даних, забезпечуючи швидкий доступ та пошук.

4. Потокова обробка.

Потокова обробка включає аналіз потоків даних у режимі реального часу для отримання корисної аналітики. Такі фреймворки, як Apache Flink, Apache Storm та Spark Streaming, використовуються для обробки великомасштабних потоків даних з мінімальною затримкою. Ці інструменти дозволяють виконувати складну обробку подій, агрегацію, об'єднання та операції вікон.

5. Пункт призначення.

Останнім компонентом є пункт призначення, куди доставляються оброблені дані. Це може бути сховище даних, база даних, озеро даних або програми кінцевих користувачів. Системи призначення зберігають кінцеві оброблені дані або запускають дії, такі як системи сповіщень, панелі моніторингу або автоматизовані робочі процеси.

Найбільш загальна схема стрімінгового сервісу показана на рисунку 1.

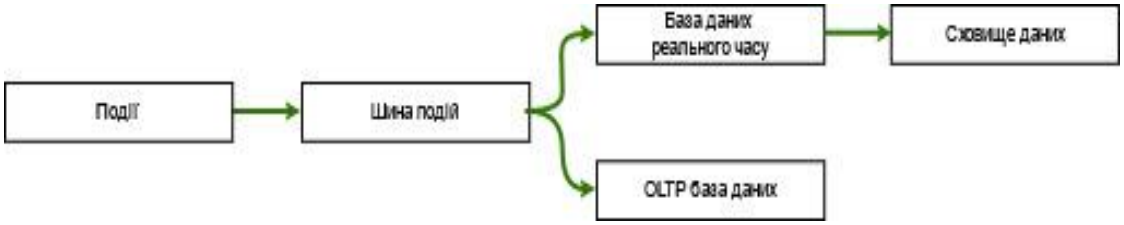

Рисунок 1. Типова схема найпростішого стрімінгового сервісу

Ця схема відображає базову архітектуру стрімінгового сервісу реального часу та ілюструє шлях обробки даних від джерела до споживача. В даному випадку споживачами є сховище даних і/або OLTP-база даних, хоча перелік таких споживачів може бути набагато ширший. Наприклад, це може бути платформа демонстрації медіа-контенту з можливостями ведення прямих ефірів, чат-бот [6], система розпізнавання образів [4] або система контролю за дорожним рухом [2].

Процес починається з події, яка надходить до шини подій – центрального компонента архітектури, що виконує роль посередника для розподілу потоків даних. Від шини подій дані розгалужуються на три напрямки: перший веде до бази даних реального часу з подальшим зберіганням у сховищі даних, другий спрямовує інформацію до OLTP бази даних для обробки транзакційних операцій, а третій забезпечує альтернативний шлях обробки.

Така архітектура забезпечує паралельну обробку подій у реальному часі, розділяючи функції оперативного зберігання, роботи з актуальними даними та довготривалого архівування, що є типовим підходом для стрімінгових платформ.

Перерахуємо кілька способів ефективного впровадження потокової передачі даних у реальному.

1. Розділення даних на розділи для паралельної обробки

Паралельна обробка має вирішальне значення для масштабування рішень потокової передачі даних у реальному часі. Розділення даних на розділи дозволяє кільком процесам одночасно обробляти різні сегменти даних, підвищуючи пропускну здатність та зменшуючи затримку. Такі технології, як Apache Kafka, використовують розділи для ефективного керування вхідними потоками даних.

2. Додавання більшої кількості вузлів до системи для обробки збільшеного навантаження.

Масштабованість є важливою для потокової передачі даних у реальному часі, і додавання більшої кількості вузлів до системи є поширеною практикою для врахування збільшених навантажень. Кожен додатковий вузол може обробляти підмножину потоку даних, розподіляючи робоче навантаження на обробку та підвищуючи загальну ємність системи. Динамічна масштабованість дозволяє системі реагувати на різні навантаження даних без шкоди для продуктивності. Впровадження масштабованої архітектури гарантує, що потокове передавання даних у реальному часі залишається ефективним та швидким у сценаріях високого навантаження, сприяючи безперебійній обробці та аналізу даних.

3. Оптимізація мережевих протоколів та логіки обробки для зменшення затримки.

Мінімізація затримки є критично важливою для потокового передавання даних у реальному часі. Оптимізація мережевих протоколів та логіки обробки може значно зменшити час затримки. Такі методи, як мінімізація накладних витрат на серіалізацію/десеріалізацію даних та використання ефективних протоколів передачі даних, сприяють зниженню затримки. Ефективна логіка обробки включає оптимізацію алгоритмів та використання обчислень у пам'яті для пришвидшення обробки даних. Ці оптимізації покращують можливості системи в реальному часі, забезпечуючи своєчасний та точний аналіз даних та прийняття рішень.

4. Налаштування розподілу ресурсів для оптимізації продуктивності.

Динамічний розподіл ресурсів гарантує, що ресурси системи відповідають поточному навантаженню даних, оптимізуючи продуктивність та економічну ефективність. Такі методи, як автоматичне масштабування, дозволяють системі автоматично регулювати обчислювальну потужність на основі попиту в реальному часі, забезпечуючи оптимальну продуктивність. Стратегії розподілу ресурсів включають

моніторинг системних показників та налаштування ресурсів процесора, пам'яті та сховища для задоволення вимог обробки. Ця адаптивність допомагає підтримувати стабільність та продуктивність системи під час пікових навантажень, підвищуючи надійність та ефективність потокової передачі даних у реальному часі.

5. Зберігання інформації про стан для обробки складних подій.

Обробка з урахуванням стану є важливою для обробки складних подій у потоках даних у реальному часі. Зберігання інформації про стан дозволяє системі відстежувати поточні події, керувати даними сеансів та співвідносити події з часом. Такі фреймворки, як Apache Flink, надають надійні можливості управління станом для підтримки цих завдань. Ефективне управління станом забезпечує точну та своєчасну аналітику, дозволяючи системі обробляти складну логіку обробки подій. Завдяки обробці з урахуванням стану, рішення для потокової передачі даних у реальному часі можуть впроваджувати розширену аналітику, забезпечуючи прийняття обґрунтованих рішень на основі безперервних потоків даних.

6. Налаштування сповіщень про незвичайну поведінку або погіршення продуктивності.

Налаштування сповіщень про незвичайну поведінку або погіршення продуктивності має вирішальне значення для підтримки справності та надійності систем потокової передачі даних у реальному часі. Моніторинг ключових показників продуктивності, таких як затримка обробки, коефіцієнти помилок та завантаження системи, дозволяє швидко виявляти проблеми. Автоматизоване сповіщення дозволяє негайно реагувати на аномалії, зменшуючи час простою та зменшуючи ризики. Та підвищуючи надійність системи.

Здатність обробляти та аналізувати дані в режимі реального часу стала критично важливою для бізнесу, дозволяючи їм отримувати, обробляти та реагувати на дані безперервно та масштабовано. Типово системи опрацювання даних в реальному часі базуються на технологіях Apache Sparks та релевантних сервісах. В рамках проведення подальших досліджень буде деталізована архітектура рішень для різних предметних областей та з використанням відповідних архітектурних програмних рішень.

Література

1. Ланевич Т. Apache Kafka та Rabbitmq: порівняння архітектур та можливостей. Тези XIII МНПК „*Актуальні задачі сучасних технологій*“, 11-12 грудня 2024 року. Тернопіль. : ФОП Паляниця В. А., 2024. С. 402–403.
2. Мадяк О. П. Використання даних про дорожній рух у реальному часі для керування дорожньо-транспортною системою. Збірник тез доповідей VI Міжнародної науково-технічної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“, 16-17 листопада 2017 року. – Т. : ТНТУ, 2017. – Том 2. – С. 108.
3. Остапчук О. Цуприк Г. Технічні особливості взаємодії між клієнтом та сервером у реальному часі. Матеріали X науково-технічної конференції „Інформаційні моделі, системи та технології“, 7–8 грудня 2022 року. – Т. : ТНТУ, 2022. С. 124. – (Програмна інженерія та моделювання складних розподілених систем).
4. Панчишин П. С. Підвищення якості розпізнавання об’єктів у відеопотоці в реальному часі / П. С. Панчишин, Михайло Іванович Паламар // Матеріали XII НТК „ІМСТ“, 18-19 грудня 2024 року. – Т. : ТНТУ, 2024. – С. 186–187.
5. Федорович І. Використання Spark Streaming та Spark Structured Streaming для обробки даних в реальному часі / Ілля Федорович, Галина Осухівська // ІМСТТ, 13-14 грудня 2023 року. Тернопіль : ТНТУ, 2023. С. 225.
6. Хом’як А. С. Підвищення ефективності обробки в реальному часі у службах чат-ботів через інтеграцію черги запитів для розподілення навантаження / А.

С. Хом’як // Матеріали XII Міжнародної науково-практичної конференції молодих учених та студентів „Актуальні задачі сучасних технологій“, 6-7 грудня 2023 року. Тернопіль : ФОП Паляниця В. А., 2023. С. 408–409.

УДК 004.415
П.М. Семчишин, студент гр. СНм-61
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

АРХІТЕКТУРНІ РІШЕННЯ ДЛЯ РОЗРОБКИ ВЕБ-ЗАСТОСУНКІВ

P.M. Semchyshyn

ARCHITECTURAL SOLUTIONS FOR WEB APPLICATION DEVELOPMENT

Коли йдеться про проектування та розробку веб-застосунків, при виборі архітектурного рішення є два основні варіанти – монолітна або мікросервісна архітектури.

Монолітна архітектура в веб-застосунках є підходом, при якому весь функціонал програми виконується в одному цілому, зазвичай у вигляді єдиного виконуваного файлу або програми [1]. Основні компоненти програми, такі як інтерфейс користувача, бізнес-логіка та доступ до даних, містяться всередині однієї програми.

Перевагами такого виду архітектури є:

– простота розгортання - оскільки всі компоненти знаходяться в одному місці, розгортання програми, зазвичай, відбувається швидше та простіше;

– простота розробки - відсутність складної інфраструктури полегшує процес розробки та налагодження програми;

– зручність масштабування на початку - при невеликому обсязі трафіку користувача монолітний застосунок може забезпечити достатню продуктивність без необхідності поділу на окремі сервіси.

Недоліки монолітної архітектури:

– складність підтримки - при збільшенні розміру програми та додаванні нових функцій може виникнути складність у підтримці та зміні коду через його єдиний монолітний характер;

– обмежена масштабованість - при досягненні межі продуктивності або необхідності масштабування окремих компонентів програми можуть виникнути труднощі через їх взаємозв'язок усередині моноліту.

– обмежена гнучкість - через єдиний характер застосування зміни в одній його частині можуть торкнутися інші компоненти, що ускладнює зміни та впровадження нових технологій.

Монолітна архітектура може бути хорошим вибором для простих застосунків з невисоким навантаженням та невеликими вимогами до масштабованості та гнучкості [1].

Мікросервісна архітектура у веб-застосунках є підходом, у якому застосунок розбивається на невеликі автономні сервіси, кожен із яких відповідає окремому функціоналу. Кожен сервіс, зазвичай, розгортається та масштабується незалежно [2].

Переваги такої архітектури:

– гнучкість та масштабованість - завдяки поділу програми на незалежні сервіси, кожен з них можна масштабувати та оновлювати незалежно від інших компонентів, що покращує гнучкість та масштабованість всієї програми;