Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Проєкт рекомендаційної системи фільмів на основі архітектури Netflix

Виконав(ла): студент(ка) 6 курсу, групи СНм-61
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

		Доберчак О.В.
	(підпис)	(прізвище та ініціали)
Керівник		Марценко С.В.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Никитюк В.В.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Боднарчук І.О.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
"17" листопада 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Доберчак Олексій Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи **Проєкт рекомендаційної системи для онлайн стрімінгових платформ.**

Керівник роботи **к.т.н., доц. Марценко С.В.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від "27" листопада 2025 року № 4/7-1042

2. Термін подання студентом завершеної роботи 21 грудня 2025 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП; 1 ОГЛЯД ПЛАТФОРМИ NETFLIX; 1.1 Особливості прямих трансляцій на стрімінгових платформах; 1.2 Спеціалізовані засоби мовлення для отримання прямого ефірного контенту; 1.3 Оптимізація відтворення в реальному часі для сумісності з пристроями, масштабування, якості та стабільності; 2 АРХІТЕКТУРА СТРІМІНГОВОЇ ПЛАТФОРМИ ДЛЯ ПРЯМИХ ТРАНСЛЯЦІЙ; 2.1 Огляд вимог до побудови архітектури системи; 2.2 Кодування та створення бітрейтів розподілу в хмарі; 2.3 Огляд елементу архітектури прямих трансляцій Live Origin; 2.4 Управління сервісами прямої трансляції; 3 ОПИС ПРОЄКТУ ПРОПОНОВАНОЇ СИСТЕМИ; 3.1 Обмеження для прямої трансляції та їх оптимізація; 3.2 Детальний аналіз роботи системи; 3.3 Приклад простої стрімінгової системи на основі Apache Kafka; 4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ; ВИСНОВКИ; СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ; ДОДАТКИ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., к.т.н., доц. каф. МТ		
Безпека в надзвичайних ситуаціях	Теслюк В.М., проректор з адміністративно-господарської роботи та будівництва		

7. Дата видачі завдання 17 листопада 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	17.11.25-18.11.25	Виконано
2.	Підбір наукових джерел за темою роботи	19.11.25-20.11.25	Виконано
3.	Переклад та опрацювання наукових джерел	20.11.25-23.11.25	Виконано
	за темою кваліфікаційної роботи		
4.	Виконання дослідження щодо теми	24.11.25-10.12.25	Виконано
	кваліфікаційної роботи		
5.	Оформлення першого розділу	11.12.25-12.10.25	Виконано
6.	Оформлення другого розділу	12.12.25-13.12.25	Виконано
7.	Оформлення третього розділу	13.12.25-14.12.25	Виконано
8.	Виконання завдання до підрозділу "Охорона праці"	08.12.25-09.11.25	Виконано
9.	Виконання завдання до підрозділу "Безпека в	10.12.25-12.12.25	Виконано
	надзвичайних ситуаціях"		
10.	Оформлення кваліфікаційної роботи	12.12.25-31.12.25	Виконано
11.	Нормоконтроль	14.12.25-15.12.25	Виконано
12.	Перевірка на плагіат	15.12.25	Виконано
13.	Попередній захист кваліфікаційної роботи	18.12.25	Виконано
14.	Захист кваліфікаційної роботи	22.12.2025	

Студент _____ Доберчак О.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марценко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

"Розробка рекомендаційної системи для онлайн-стрімінгових платформ" // Кваліфікаційна робота освітнього рівня "Магістр" // Доберчак Олексій Володимирович // Тернопільський національний технічний університет ім. І. Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2025 // с. – 47, рис. – 13, табл. – 1, джерел – 17.

Ключові слова: рекомендаційна система, стрімінгова платформа, прямий ефір, персоналізація, алгоритми

Відеоконтент надається споживачам за допомогою мереж доставки контенту (CDN). Зазвичай відео транслюється на вимогу, але прямі трансляції здійснюються все частіше (наприклад, відеоконтент у прямому ефірі в YouTube Live). Для забезпечення кінцевим користувачам відеопотоки високої якості, оригінальне відео має бути перекодовано для доставки через CDN.

Архітектура програмного забезпечення для ведення прямих ефірів з прогнозуванням віртуальних ресурсів для рекомендацій контенту та для транскодування відео в реальному часі може здійснюватись на основі платформи Kubernetes чи власного програмного забезпечення, як у Netflix, наприклад.

ANNOTATION

"Development of a Recommendation System for Online Streaming Platforms" // Master's degree qualification paper // Doberchak Oleksii // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, group СНм-61 // Ternopil, 2025 // p. – 47, fig. – 13, tables – 1, references – 17.

Key words: recommendation system, streaming platform, live broadcast, personalization, algorithms

Video content is delivered to consumers via content delivery networks (CDNs). Video is typically delivered on-demand, but live streaming is becoming increasingly common (e.g., live video content on YouTube Live). To provide high-quality video streams to end users, the original video must be transcoded for delivery via the CDN.

The architecture of a live streaming software with predictive virtual resources for content recommendations and real-time video transcoding can be built on Kubernetes or on proprietary software, such as Netflix.

ЗМІСТ

ВСТУП

Актуальність задачі.

На відміну від відео на вимогу (VOD), учасники хочуть бачити події в прямому ефірі одразу після їх початку. Це означає, що у є лише короткий відтинок часу, щоб порекомендувати подію в прямому ефірі саме в потрібний час.

Щоб відповідати цій вимозі, потрібно вдосконалити системи доставки рекомендацій, щоб вони надавали пропозиції в режимі реального часу, надаючи учасникам більш змістовні та переконливі сигнали для відтворення контенту в той момент, коли це найважливіше. Це досягається архітектурними рішеннями, використанням спеціалізованих платформ обробки даних в реальному часі та алгоритмами машинного навчання.

Мета роботи.

Метою роботи є дослідження можливостей надсилати клієнтам динамічні, своєчасні оновлення одночасно на сотні тисяч чи більше пристроїв, не створюючи ефекту "масового натовпу", який би перевантажив хмарні сервіси. Просте лінійне масштабування не є ефективним та надійним. Для популярних подій це також може відволікати ресурси від інших критично важливих сервісів.

Для досягнення цієї мети варто вирішити такі задачі:

1. Дослідити механізми технічної реалізації прямих трансляцій на стрімінгових платформах.
2. Запропонувати архітектурне рішення для надання рекомендацій на стрімінгових платформах для прямих трансляцій.
3. Дослідити вплив алгоритмів та платформ кодування відеосигналу для прямих трансляцій.

Об’єкт дослідження: процеси розробки архітектури рекомендаційної системи онлайн-трансляцій для стрімінгової платформи.

Предмет дослідження: системи рекомендацій для платформи прямої трансляції відеоконтенту.

Наукова новизна отриманих результатів.

Проведено огляд предметної області та літературних джерел, щоб знайти фактори, які впливають на якість онлайн-трансляції з боку алгоритмів транскодування та надання рекомендацій на прикладі архітектури Netflix. Отримані рішення можна використати в інших стрімінгових сервісах.

Практичне значення отриманих результатів.

У даній роботі отримано результати, що мають практичне значення для використання при розробці платформ онлайн-трансляцій в прямому ефірі для покращення рекомендацій та якості відеосигналу.

Апробація результатів та особистий внесок здобувача.

Основні положення роботи доповідались, розглядались та обговорювались на науковій конференції Тернопільського національного технічного університету імені Івана Пулюя. Результати кваліфікаційної роботи опубліковані у тезах студентської наукової конференції "Актуальні задачі сучасних технологій – 2025", яка проводилась у ТНТУ.

1 ОГЛЯД ПЛАТФОРМИ NETFLIX

1.1 Особливості прямих трансляцій на стрімінгових платформах

Серед контенту стрімінгових платформ на зразок Netflix, SweetTV, YouTube та інших все більшого розповсюдження набувають прямі трансляції з різних заходів, подій. Таке поширення цього типу контенту вимагає нових рішень і нових підходів до технічної реалізації та до програмного забезпечення для трансляції.

Хоча телевізійний формат Live не є новим, потокове передавання вимагало можливостей, що впливають на вибір архітектури та технологій (див. рис. 1.1).

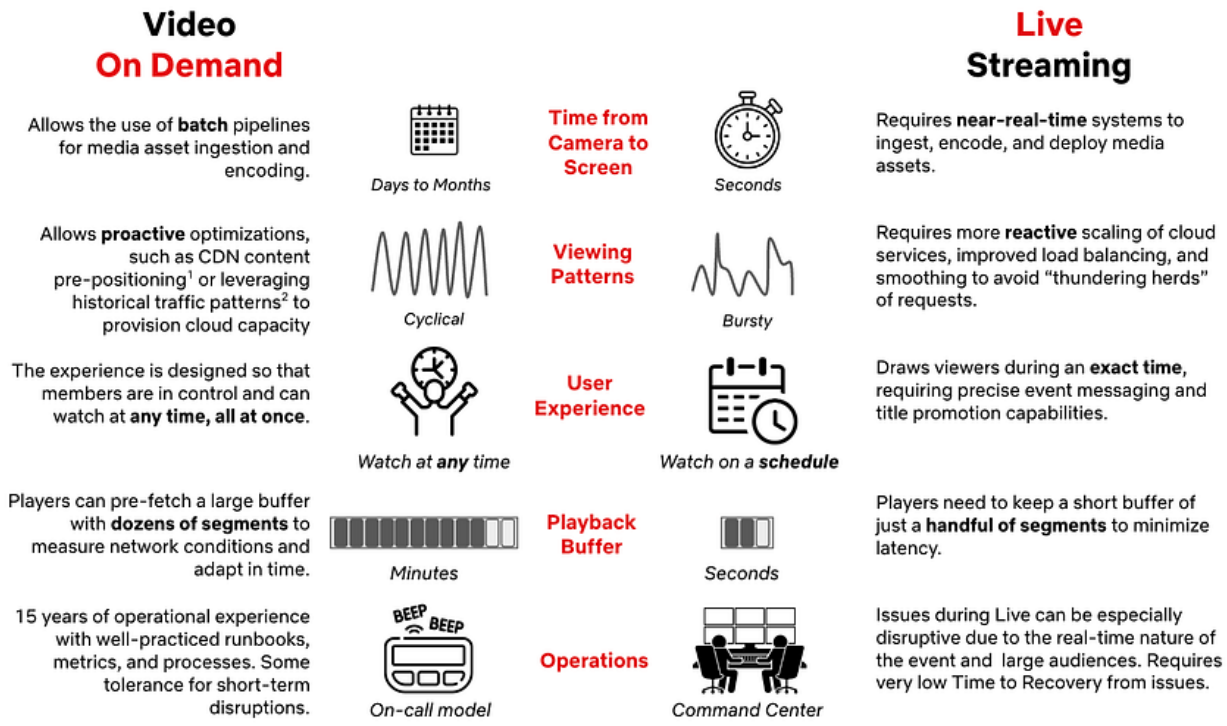

Рисунок 1.1 – Перелік технологій для прямих трансляцій

Такий перелік означає, що ці особливості потребують великого переліку рішень, які мають бути реалізовані в систему прямої трансляції з підтримкою рекомендацій.

Технології прямих трансляцій мали надати користувачам ту саму можливість, яку надається їм для потокової трансляції на вимогу: чудова якість

на якомога більшій кількості пристроїв без перерв. Прямі трансляції – це один із багатьох розважальних форматів, тому також потрібно інтегрувати прямі трансляції в користувацький досвід, одночасно масштабуючись до величезної кількості користувачів по всьому світу (до сотень мільйонів для Netflix).

Загальний вигляд архітектури такої системи показано на рисунку 1.2.

1.2 Спеціалізовані засоби мовлення для отримання прямого ефірного контенту з виробництва

Прямі трансляції можуть відбуватися будь-де у світі, але не в кожному місці є обладнання для прямих трансляцій або високоякісне з'єднання. Щоб забезпечити безпечну та надійну передачу сигналу в прямому ефірі, використовуються розподілені та високорівневі центри операцій мовлення зі спеціалізованим обладнанням для прийому та перевірки сигналу, субтитрів, управління графікою та рекламою. Пріоритет надається повторюваності, зосереджуючи інфраструктуру для послідовного, надійного та економічно ефективного запуску прямих трансляцій, спираючись на автоматизацію, де це можливо. В результаті є можливість звести налаштування для кожної події до передачі між виробництвом та центром операцій мовлення, повторно використовуючи решту для різних подій.

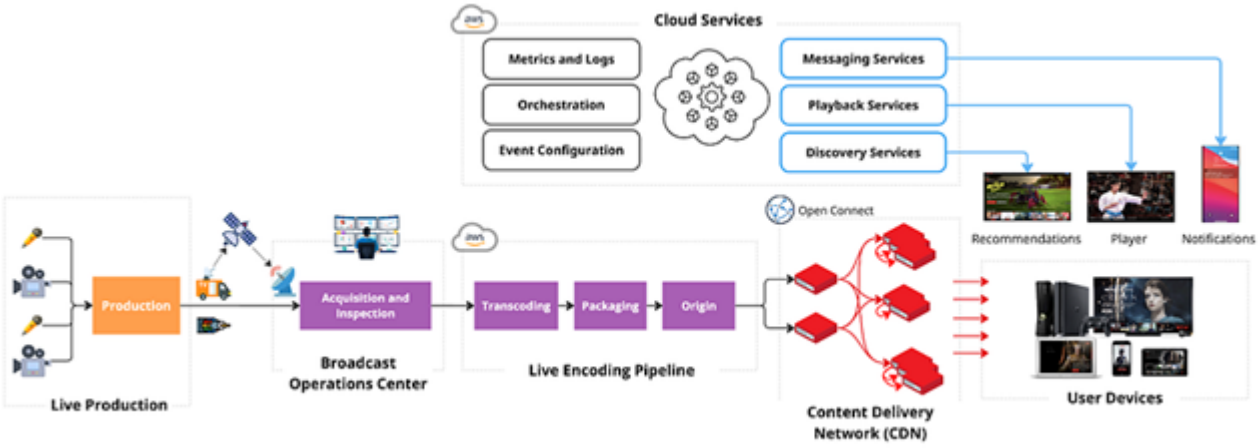

Рисунок 1.2 – Архітектура стрімінгової системи з підтримкою прямих трансляцій

Стрічка рекомендацій, отримана в трансляційному центрі, містить повністю підготовлену програму, але її все ще потрібно закодувати та упакувати для потокової передачі на пристроях. Доцільним є використання для цього хмарного підходу, щоб забезпечити динамічне масштабування, гнучкість у налаштуванні та легкість інтеграції з іншими службами управління цифровими правами (DRM), управління контентом та доставки контенту, які вже розгорнуті в хмарі. Для платформи Netflix використовуються AWS Elemental MediaConnect та AWS Elemental MediaLive для отримання стрічок у хмарі та перекодування їх у різні рівні якості відео з бітрейтом, адаптованим до кожної передачі. Також створено спеціальний пакувальник для кращої інтеграції з нашими системами доставки та відтворення. Створено також спеціальний сервіс Live Origin, щоб забезпечити дотримання вимог про рівень обслуговування на читання та запис для сегментів прямої трансляції.

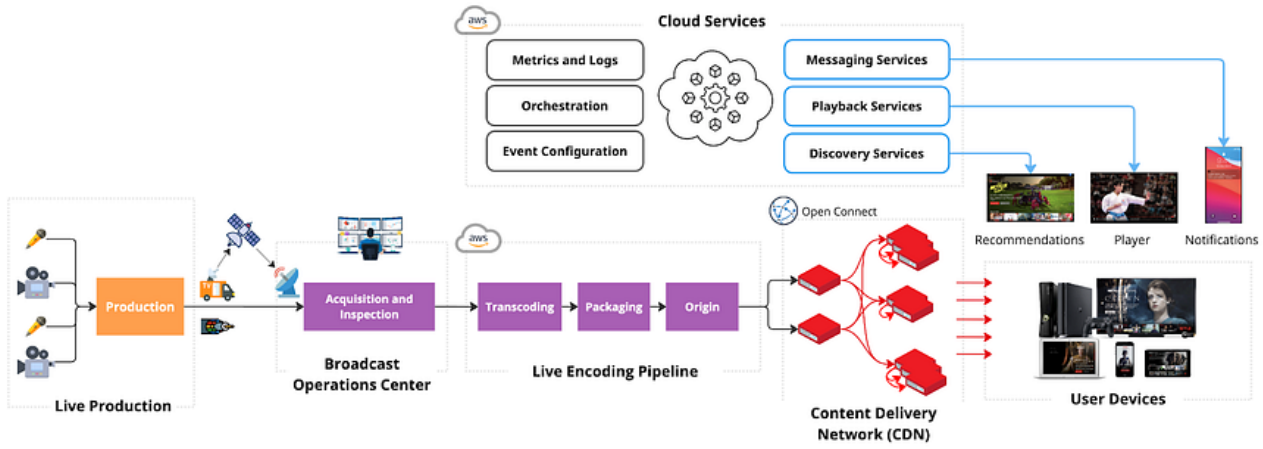

Рисунок 1.2 – Архітектура сервісу з функціями прямої трансляції

Для потокової передачі створених медіаресурсів їх потрібно перенести з кількох локацій AWS, де розгорнуто Live Origin, на всю наявну кількість пристроїв по всьому світу. Для цього використовується CDN Netflix, Open Connect [3], для масштабування доставки ресурсів Live. Сервери Open Connect розміщені поблизу глядачів у понад 6 тисячах локацій та підключені до локацій AWS через виділену магістральну мережу Open Connect (див. рис. 1.3).

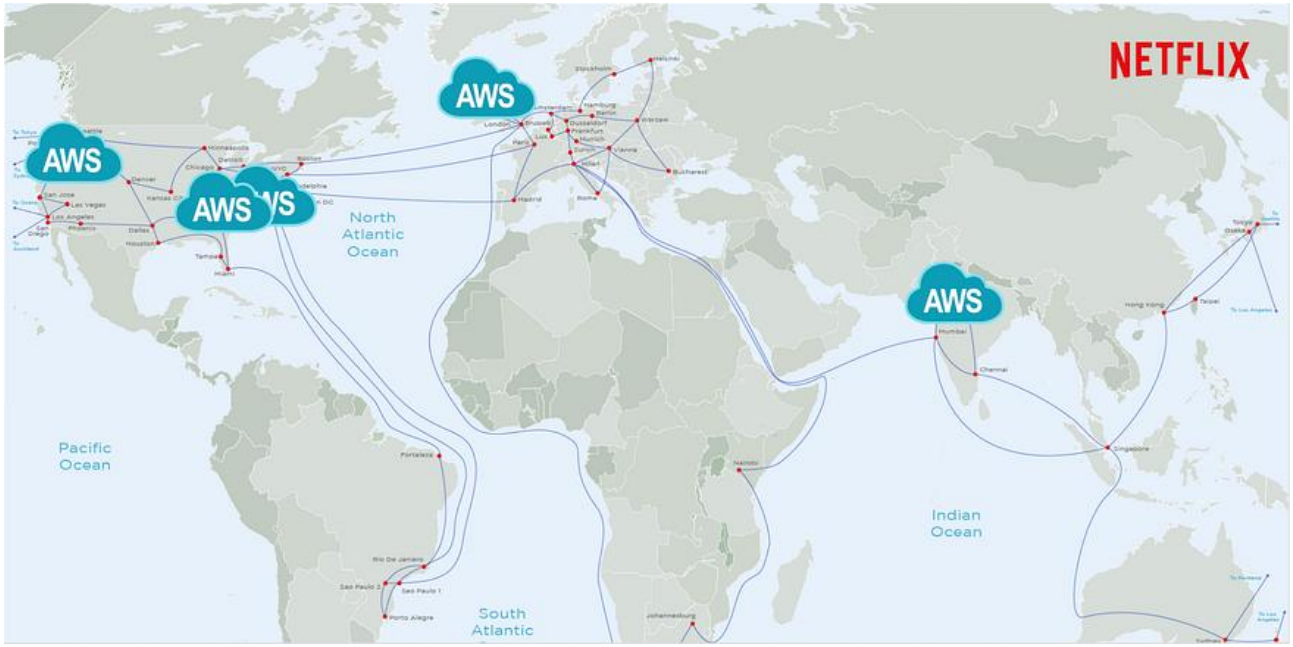

Рисунок 1.3 – Open Connect Backbone з'єднує сервери в місцях розташування Internet Exchange з 5 регіонами AWS [4]

Завдяки можливості прямої трансляції на Open Connect є можливість для масштабування мережі та оптимізацію продуктивності серверів доставки. Розподіляючи потужності між глядачами на вимогу та в прямому ефірі, покращується їх використання, а кешуючи попередній контент у прямому ефірі на тих самих серверах, що використовуються для потокової трансляції на вимогу, можна легко забезпечити перегляд з затримкою в часі.

1.3 Оптимізація відтворення в реальному часі для сумісності з пристроями, масштабування, якості та стабільності

Щоб зробити Live доступним для більшості клієнтів без оновлення їхніх потокових пристроїв використовується пряма трансляція на основі HTTPS. Хоча протоколи на основі UDP можуть надавати додаткові функції, такі як наднизька затримка, HTTPS має повсюдну підтримку серед пристроїв та сумісний із системами доставки та кодування. Крім того, застосовуються відеокодеки AVC та HEVC, перекодування з кількома рівнями якості від SD до 4K та використання 2-секундного сегменту для балансування ефективності стиснення, навантаження

на інфраструктуру та затримки. Надаючи пріоритет якості потокової передачі та стабільності відтворення, також досягнуто галузевого стандарту затримки від камери до пристрою.

Щоб налаштувати відтворення, програвач пристрою отримує маніфест відтворення на початку відтворення. Маніфест містить такі елементи, як бітрейт кодування та сервери CDN, які повинні використовувати програвачі. Маніфест доставляється із хмари, а не з CDN, оскільки це дозволяє персоналізувати конфігурацію для кожного пристрою. Для посилання на сегменти потоку маніфест містить шаблон сегмента, який використовується пристроями для зіставлення часу настінного годинника з URL-адресами на CDN. Використання шаблону сегмента замість періодичного опитування для оновлень маніфесту мінімізує мережеві залежності, навантаження на сервер CDN та накладні витрати на пристрої з обмеженими ресурсами, такі як смарт-телевізори, тим самим покращуючи масштабованість і стабільність системи. Під час потокової передачі програвач контролює продуктивність мережі та динамічно вибирає бітрейт і сервер CDN, максимізуючи якість потокової передачі, мінімізуючи перебуферизацію.

Досі шлях потокової передачі розглядався від камери до пристрою. Щоб трансляція працювала повноцінно, також потрібно оркеструвати всі системи та забезпечити, щоб глядачі могли знайти та розпочати пряму трансляцію. Цю функціональність виконують десятки хмарних сервісів, з такими функціями, як налаштування відтворення, персоналізація або збір метрик. Ці сервіси, як правило, отримують непропорційно більше навантаження приблизно під час початку прямої трансляції, а хмарне розгортання забезпечує гнучкість у динамічному масштабуванні обчислювальних ресурсів. Більше того, оскільки попит на пряму трансляцію, як правило, локалізований, то можна балансувати навантаження між кількома регіонами AWS, краще використовуючи глобальний вплив. Розгортання в хмарі також дозволяє створювати користувацький досвід, де пряма трансляція вбудовується в ширший вибір розважальних опцій в інтерфейсі користувача, таких як заголовки на вимогу або ігри.

Завдяки контролю над завантаженням, конвеєрами кодування, Open Connect CDN та програвачами пристроїв, наявний майже повний контроль над робочим процесом Live. Під час Live збираються системні та користувацькі показники в режимі реального часу (наприклад, де учасники бачать заголовок якогось ресурсу та якість їхнього досвіду), що попереджає про поганий користувацький досвід або погіршення продуктивності системи. Моніторинг у режимі реального часу побудовано з використанням поєднання внутрішньо розроблених інструментів, таких як Atlas, Mantis та Lumen, та технологій з відкритим кодом, таких як Kafka та Druid , обробляючи до 38 мільйонів подій на секунду під час деяких з наших найбільших прямих трансляцій, надаючи критичні показники та операційну аналітику за лічені секунди. Крім того, створено спеціальні об'єкти «Центр керування», які об'єднують ключові показники для операційної команди, яка контролює подію в режимі реального часу.

Створення нового функціоналу завжди створює нові виклики та можливості для навчання, особливо з такою складною системою, як Live. Ось кілька ключових моментів, необхідних для налагодження якісної прямої трансляції.

Ретельне тестування. До запуску Live ми значною мірою покладалися на передбачуваний потік трафіку на вимогу для попередніх тестів або A/B-тестів для перевірки розгортань. Але Live-трафік не завжди був доступний, особливо не в масштабах, характерних для великого запуску. Як результат, ми доклали значних зусиль, щоб:

1. Створити внутрішні «тестові потоки», які інженери використовують для проведення інтеграційних, регресійних або димових тестів як частину життєвого циклу розробки.

2. Створити можливості синтетичного навантажувального тестування для стрес-тестування хмарних систем та систем CDN. Пропонується від Netflix 2 підходи, що дозволяють генерувати до 100 тисяч запусків за секунду:

– Захоплення, зміна та відтворення минулого трафіку Live production, що представляє різноманітні пристрої користувачів та шаблони запитів.

– Віртуалізація пристроїв Netflix та генерація трафіку для кінцевих точок CDN або хмари, щоб перевірити вплив останніх змін на всі системи.

3. Запускати автоматичне впровадження помилок, примусово додаючи відсутні або пошкоджені сегменти з конвеєра кодування, втрату хмарної області, падіння мережі або перевищення часу очікування сервера.

Регулярна практика/ Незважаючи на ретельне передрелізне тестування, ніщо не зрівняється з виробничим середовищем, особливо під час роботи у великих масштабах. Регулярний графік із різноманітним прямим ефіром є важливим для вдосконалення, одночасно збалансовуючи ризики впливу на учасників. Проводиться A/B-тестування (сегментне), операційні вправи та навчання операційних команд майбутнім запускам.

Прогнози щодо кількості переглядів. Використовуються методи на основі прогнозування для попереднього виділення потужностей хмарних сервісів та CDN, а також заздалегідь відбувається надання прогнозів інтернет-провайдерам та хмарним партнерам, щоб вони могли планувати мережеві та обчислювальні ресурси. Потім вони доповнюються реактивним масштабуванням хмарних систем, що забезпечують реєстрацію, вхід, пошук заголовків та відтворення, щоб врахувати перевищення прогнозів щодо кількості переглядів. Важливим є досягнення успіху з прогнозами щодо кількості переглядів у реальному часі під час прямої трансляції, що дозволяє вживати заходів для зменшення ризиків раніше, до того, як це вплине на більшу кількість учасників.

Незначна деградація сервісу. Незважаючи на всі зусилля, можлива ситуація, коли кількість глядачів перевищить прогнози та виділену потужність. У цьому випадку існує ряд засобів для продовження потокового передавання, навіть якщо це означає поступове видалення деяких корисних функцій. Наприклад, використовується пріоритетне розвантаження на рівні сервісу, щоб надати пріоритет живому трафіку над некритичним трафіком (наприклад, попередня вибірка). Крім того, можна спростити враження, наприклад,

зменшити персоналізацію, вимкнути закладки або знизити максимальну якість потокового передавання.

Навантаження повторних спроб. Коли система досягає максимальних можливостей, то головною метою є уникнення каскадних проблем або подальшого перевантаження систем повторними спробами. Окрім системних повторних спроб, користувачі можуть повторювати спроби вручну – тоді можливе 10-кратне збільшення навантаження на трафік через перезапуск потоку після перерв тривалістю всього 30 секунд.

Планування на випадок надзвичайних ситуацій. Коли щось ламається, практично немає часу на усунення несправностей. Для швидкого виявлення та реагування розроблено невеликий набір показників як ранні індикатори проблем і наявні розширені методичні рекомендації для поширених операційних проблем. Важливо не навчатися в день запуску; натомість команди запуску заздалегідь відпрацьовують реагування на збої за допомогою імітацій.

2 АРХІТЕКТУРА СТРІМІНГОВОЇ ПЛАТФОРМИ ДЛЯ ПРЯМИХ ТРАНСЛЯЦІЙ

2.1 Огляд вимог до побудови архітектури системи

Коли Netflix почав досліджувати можливості прямого мовлення, метою було забезпечити прямий ефір, який міг би охопити якомога більше пристроїв, що підтримують платформу, і відповідати або перевищувати галузеві стандарти якості та затримки. Однак шлях до досягнення цієї мети був аж ніяк не простим, оскільки вимагав балансу між швидкістю виконання, надійністю та широкою сумісністю пристроїв з першого дня.

Хоча учасники з різних функціональних груп були віддані ідеї трансляції подій у прямому ефірі, для членів команди залишалося ще багато невідомого. Ці труднощі посилювалися відсутністю існуючої інфраструктури для трансляції, перевірки, обробки та доставки прямих трансляцій. Тому потрібно було створити базові можливості з нуля, включаючи розробку надійних методів передачі сигналів у прямому ефірі з місць проведення заходів до хмари.

Щоб впоратися з цими невідомими, було пріоритезовано рішення, які вже були частиною прокладеного шляху Netflix або були знайомі командам розробки. Основні зусилля були зосереджені на створенні надійного та гнучкого конвеєра, який міг би адаптуватися в міру розвитку прямих трансляцій. Такий підхід дозволив експериментувати, проводити ітерації та зрештою закласти основу для проведення високоякісних прямих трансляцій для своєї аудиторії.

Одним із перших вирішальних рішень було те, як діяти без існуючої інфраструктури мовлення. Не було чіткого розуміння того, як і куди будуть доставлятися сигнали прямих трансляцій. Було прийнято рішення побудувати можливості збору та обробки даних безпосередньо в хмарі, виходячи з припущення, що прямі трансляції сигналів з шоу певним чином будуть доставлятися в це хмарне середовище. Глибокі знання протоколів передачі та глибоке розуміння їхніх можливостей надійності вплинули на початковий підхід,

прагматичний вибір, який дозволив швидко діяти та підтримувати стандарти надійності, яких очікують користувачі сервісу.

Для першого шоу в прямому ефірі було створено підключення на місці, щоб забезпечити бездоганну трансляцію. Щоб дістатися до хмарних кодерів, було налаштовано два різні виділені канали доступу до Інтернету (DIA) з місця проведення, надсилаючи UHD-канали зі швидкістю 50 Мбіт/с за допомогою HEVC, щоб максимізувати якість у межах бюджету на пропускну здатність (див. рис. 2.1).

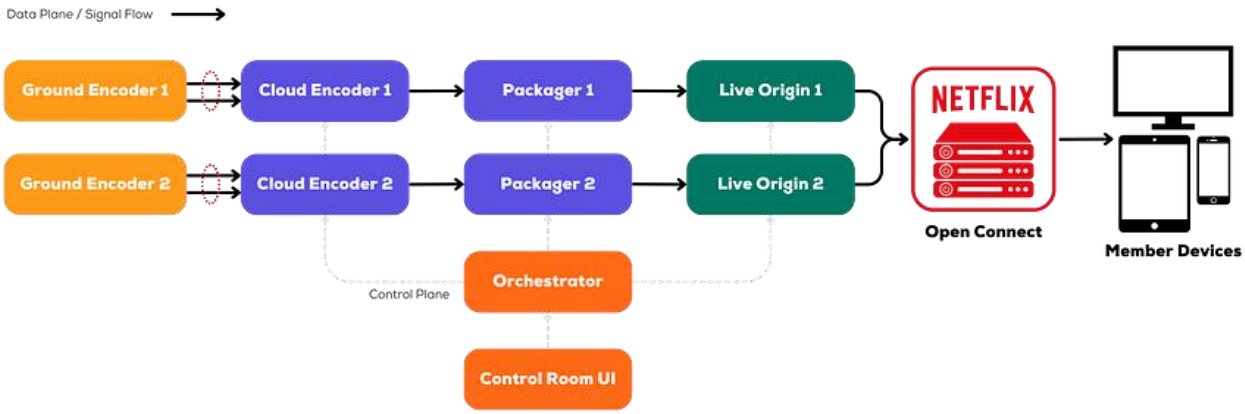

Рисунок 2.1 – Використання двох виділених каналів для прямої трансляції

Хоча цей прямий шлях від місця проведення до хмари забезпечив перші кілька шоу, він виявив обмеження масштабованості. Стало зрозуміло, що зростання вимагає позбавлення впливу від мінливості місць проведення на користь більш послідовного підходу. Тоді застосовувалвсь модель «хаб і спиці», яка агрегує виробничі потоки на добре підключеному телевізійному центрі перед їх передачею в хмару.

Ще однією частиною стратегії хмарного інгестування є забезпечення кількох рівнів резервування та усунення окремих точок відмови. Хмарне інгестування працює у двох окремих регіонах, кожен з яких отримує канали через два окремі керовані прямі мережеві шляхи, а не через відкритий Інтернет. Така схема передає прямий ефір у вигляді чотирьох незалежних потоків, забезпечуючи кілька рівнів захисту від мережевих проблем, збоїв або відмов обладнання. В основі цього резервування лежить безперебійна комутація захисту

SMPTE 2022–7, яка об'єднує подвійні шляхи на рівні пакетів і автоматично вибирає найкращі пакети в режимі реального часу. Якщо на одному шляху відбувається втрата або погіршення якості пакетів, система миттєво перемикається на альтернативний шлях без будь-якого впливу на кодування нижче за потоком передачі даних. На практиці це працює бездоганно, зберігаючи якість сигналу навіть під час втрати пакетів або повних збоїв шляху. Для масштабування цієї архітектури використовується AWS Elemental MediaConnect для керованого транспортування відео, резервування та відновлення після відмови. Його інтеграція з хмарними сервісами кодування спрощує операції та надає необхідні інструменти моніторингу.

2.2 Кодування та створення бітрейтів розподілу в хмарі

Щойно сигнал трансляції надходить до хмари, він переходить на наступний етап, який називається конвеєром прямої трансляції. Стрім має бути закодований та упакований для використання клієнтом. Хмарний кодер відповідає за створення сегментованих потоків з адаптивними сходами бітрейту, забезпечуючи оптимальну якість для різних інтернет-підключень та пристроїв. Щоб забезпечити найкращу якість кодування, слід ретельно оптимізувати кроки бітрейту відео на основі роздільної здатності, частоти кадрів та складності контенту, щоб забезпечити найкращу якість відео для учасників. Для подій з високою динамікою, таких як спортивні змагання, виділяється вищий бітрейт, щоб забезпечити учасникам високоякісний досвід.

Щоб забезпечити роботу з широким спектром пристроїв, включаючи застарілі типи, які можуть не підтримувати сучасні формати кодування, кодер налаштовано на одночасне створення кількох відео- та аудіоформатів. Для відео підтримуються AVC та HEVC. Для аудіо підтримуються HE-AAC та Dolby Digital Plus 5.1 з різними бітрейтами. Крім того, аудіо транслюється кількома мовами для підтримки глобального охоплення Netflix. Для доставки прихованих субтитрів створюються як WebVTT, так і TTML/IMSC.

Така архітектура сприяє стійкій роботі системи з двома конвеєрами та забезпечує плавне перемикання між конвеєрами. Архітектура відповідає стандартизованому підходу, викладеному в ISO/IEC 23009–9, «Динамічне адаптивне потокове передавання через HTTP (DASH)» (REaP).

Ключовим аспектом конструкції REaP є те, що два конвеєри створюють взаємозамінні сегменти без прямого зв'язку компонентів у різних конвеєрах, що дозволяє вихідним або наступним компонентам вибирати між ними. Для досягнення цього налаштовується постійна середня тривалість сегмента як для кодера, так і для пакувальника. Кодер-вкладник вбудовує часовий код у кожен відеокадр за допомогою повідомлень SEI (додаткова інформація про покращення). Це передає інформацію про час UTC для кожного кадру до конвеєра хмарного кодування. Для забезпечення безперебійного перемикання хмарний кодер детерміновано зіставляє кадри з часовими кодами з вихідними сегментами.

Також було створено власний пакет-пакувальник. Це рішення дозволило найкраще використати існуючі компоненти потокової системи Netflix та зберегти сумісність потокового передавання з пристроями, що є надзвичайно важливим для дотримання термінів запуску та досягнення широкого охоплення пристроїв.

Стримінговий сервіс Netflix підтримує мільйони пристроїв із сотень категорій, постачальників, а також версій апаратного та програмного забезпечення. Забезпечення сумісності форматів потокового передавання з цими пристроями вимагало багато років цілеспрямованих зусиль, включаючи коригування потокових форматів та масштабні польові випробування. Пакувальний інструмент відповідає за сумісність бітових потоків на системному рівні. Щоб забезпечити сумісність функції прямого мовлення з пристроями, що підтримуються Netflix, без повторення трудомістких польових випробувань, використання перевіреної формули пакування SVOD-пакувального інструменту в Live-пакувальному інструменті стало найбільш життєздатним рішенням.

Окрім форматів потокового відео та аудіо, кінцеві пристрої використовують сегменти звичайного тексту, використовуючи стандарт IMSC або WebVTT. Відсутність підтримки налаштованих форматів субтитрів у сторонніх рішеннях для пакування ще більше зумовила необхідність створення власного пакувальника відеосигналу трансляції для прямої трансляції.

Зрештою, важливою функцією пакувальника є захист потоків за допомогою керування цифровими правами (DRM). Пакувальник повністю інтегрований із серверною частиною безпеки для генерації ключів шифрування, шифрування сегментів та автентифікації безпеки пакувальника. Пакувальник, що працює в режимі реального часу, може легко використовувати ці існуючі компоненти DRM, створюючи ефективні рішення та пришвидшуючи час виведення на ринок.

У першій частині цієї роботи було згадано про використання стрімінговою платформою персоналізованих маніфестів та шаблонів сегментів з постійною тривалістю сегмента 2 секунди. Цей підхід із шаблонами також адаптує персоналізований маніфест для підтримки прямих трансляцій без суттєвих змін архітектури серверних систем. Пакувальник відеосигналу прямих трансляцій має завдання генерувати метадані шаблону сегмента та надсилати їх до серверних систем для створення маніфестів, орієнтованих на клієнта.

Зрештою, пакувальник прямих трансляцій завдяки своїм функціям адаптації та сумісності потоків, захисту DRM та генерації шаблонів сегментів прямих трансляцій, забезпечує успішні сеанси прямих трансляцій, мінімізуючи архітектурні зміни, необхідні в інших частинах системи. Пакувальник прямих трансляцій також створює платформу для інновацій та закладає основу для майбутніх розширених функцій.

2.3 Огляд елементу архітектури прямих трансляцій Live Origin

Для прямих трансляцій сервери походження виступають початковим джерелом для мережі розповсюдження контенту. З низькими значеннями

затримок, масштабовані та надійні можливості зберігання та розповсюдження є важливими для успіху масштабних прямих трансляцій.

Конвеєр прямих трансляцій спочатку спирався на статичні сховища в хмарі AWS. Щоб забезпечити резервування подвійного конвеєра, було створено два контейнери – по одному в кожному регіоні AWS. Для додаткової стійкості пакувальник у кожному регіоні публікував сегменти в обох контейнерах – процес перехресної публікації. Такий підхід забезпечив чотири потенційних кандидати для кожного сегмента. Вузли CDN могли потім застосовувати власну логіку пріоритезації та резервного варіанту для вибору оптимального кандидата для доставки, що ще більше підвищувало резервування та надійність потокової передачі (див. рис. 2.2).

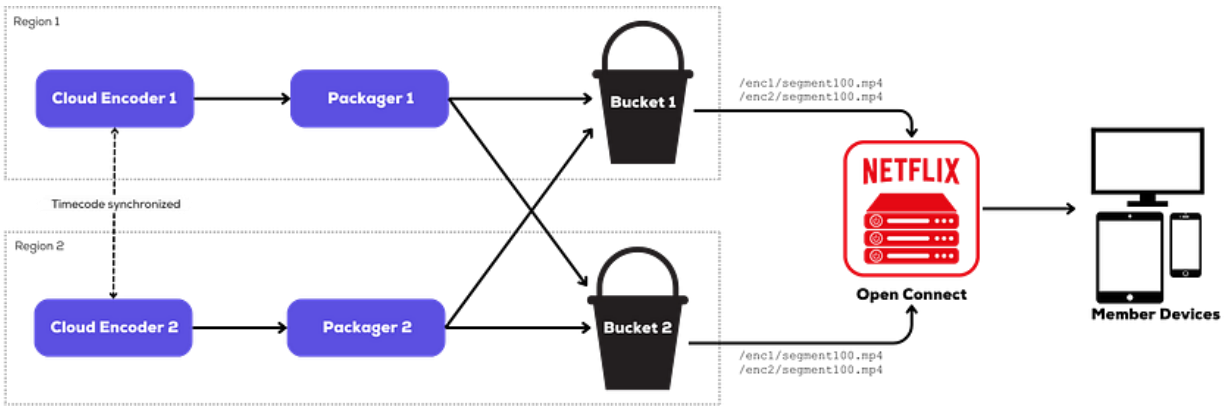

Рисунок 2.2 – Дублювання каналів упаковки початкового сигналу трансляції

Статичне сховище оригіналу (Bucket 1 і 2 на рисунку 2.2) створило деякі проблеми. Основними проблемами були продуктивність та надійність сховища оригіналу. Як універсальне рішення для зберігання, статичне корзиноподібне сховище не було оптимізовано для прямої трансляції та не мало необхідних угоди про рівень обслуговування (SLA) щодо доступності та продуктивності. Зокрема, публікація сегментів зазнавала високих коливань затримки. За постійної тривалості сегмента 2 секунди високі затримки порядку секунд значно погіршували продуктивність прямої трансляції. Крім того, через одночасні події в прямому ефірі та високу кількість запитів, що потрапляли до оригіналу на подію (перевищуючи 100 RPS), публікація сегментів часто стикалася зі збоями

прискорення сигналу. Навіть за умови подвійної публікації пакувальником, яка частково усувала проблеми з надійністю, коли було доступне одне корзиноподібне сховище, одночасні збої публікації все одно траплялися в обох регіонах.

Реалізація резервного подвійного конвеєра призводить до появи двох різних версій кожного сегмента або маніфесту. Якщо в одному конвеєрі виникає переривання в генерації сегмента або маніфесту, CDN може вирішити розповсюдити доступну альтернативу. Однак часто виникають ситуації, коли одна версія є життєздатною, а інша містить дефекти, які призводять до перебоїв відтворення клієнта. На жаль, CDN не може розрізнити двох унікальних кандидатів за допомогою статичних корзин.

Це спонукало розробити рівень оригіналу, оснащений медіа-залежними функціями, що значно покращило якість потокової передачі, дозволяючи вибирати оптимальних кандидатів з потокових конвеєрів. Інтегруючи інтелект в оригінал прямої трансляції, також відкрилися нові можливості для покращення операцій CDN для прямої трансляції, такі як впровадження розширеного керування кешем TTL та ефективне поширення метаданих потокової передачі. Ці переваги мотивували розробити сервіс Live Origin з високорівневою архітектурою, зосередженою на масштабованості, високій продуктивності, медіа-аудиторії та оптимізації для прямої трансляції. Платформа даних пропонує потужні рішення для зберігання даних, такі як платформа сховища даних ключ-значення [5]. На рисунку 2.3 показано вдосконалену архітектуру системи для прямої трансляції з вдосконаленим рішенням для зберігання даних.

2.4 Управління сервісами прямої трансляції

На початку запуску платформи прямих трансляцій було надано пріоритет автоматизації робочих процесів, де це було можливо. Для управління всіма сервісами було створено систему оркестрації, яка динамічно налаштовує

конвеєри кодування в прямому ефірі на основі конкретних вимог та специфікацій до контенту та виробництва.

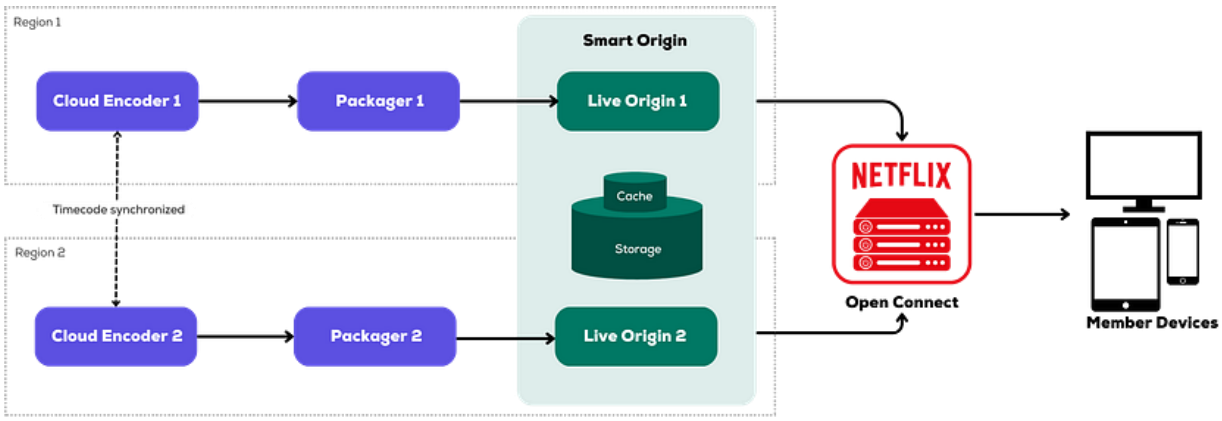

Рисунок 2.3 – Вдосконалена архітектура системи

З розширенням контенту, включивши спортивні події, реаліті-шоу та церемонії нагородження, ми використовували цей сервіс оркестрації для створення індивідуальних конвеєрів кодування для кожної події. Завдяки сервісу оркестрації можна забезпечити високоякісний досвід для учасників, мінімізуючи операційні витрати.

Хмарний конвеєр охоплює служби завантаження, кодування, пакування, відбору сигналу та моніторингу. Використовуючи систему оркестрації для налаштування та забезпечення кожного з цих компонентів, значно знижуєтьяс ризик людських помилок і дозволяє командам зосередитися на моніторингу вищого рівня, швидкому реагуванні та розробці функцій, а не на виснажливих, але критично важливих завданнях налаштування. Щоб зменшити кількість точок збою в робочому процесі, створюються надлишкові конвеєри кодування в різних географічних регіонах для кожної події. Використання служби оркестрації для цього дає гнучкість для виділення ресурсів у певних регіонах залежно від місця виробництва контенту та впевненість у тому, що всі конфігурації конвеєра ідентичні та генерують взаємозамінні медіасегменти. Служба автоматично забезпечує та скасовує екземпляри кодування на основі розкладу події в прямому ефірі, що оптимізує ресурси та, в свою чергу, витрати. Також використовується

оркестратор для запуску тестових подій з різними конфігураціями, включаючи нові сходинки бітрейту, відеокодеки та нові компоненти хмарного конвеєра, такі як, наприклад, сервіс субтитрів.

Оркестратор також керує кількома операціями в режимі реального часу, включаючи налаштування вікон перегляду в реальному часі, визначення часу переходу користувачів між подіями в реальному часі та ручні операції кодувальника, що використовуються для зменшення інцидентів.

Реалізація системи оркестрації використовує архітектурну модель типу «хаб і спиці» для створення та керування подіями в реальному часі. Центральний об'єкт керує ресурсами, дозволами та сховищем. Кожен окремий компонент, включаючи кодувальник, пакувальник, джерело тощо, представлений як спиця, яка відповідає за певний набір завдань. Сервіс керує станами конвеєра та компонентів за допомогою комбінації механізмів публікації-підписки та опитування. Для забезпечення надійності площина керування охоплює кілька географічних регіонів, причому кожне окреме регіональне розгортання здатне оркеструвати ресурси по всій глобальній інфраструктурі.

На ранніх етапах розробки сервісу керування всіма системами здійснювалось під час подій прямої трансляції. Стандартні інструменти не могли масштабуватися для таких випадків, тому запропоновано використовувати програмне рішення у вигляді інтерфейсу користувача «Кімната керування», який взаємодіяв би з рівнем оркестрації для контролю та опитування стану ресурсів подій (див. рис. 2.4).

Диспетчерська надає єдину панель керування, де оператори контролюють та керують резервною дворегіональною архітектурою. Кожна подія проходить три окремі фази: налаштування до події, кодування (включаючи пряму трансляцію) та очищення після події. Проте, все налаштування та демонтаж виконувалися вручну через диспетчерську до та після події.

Протягом вікна передачі (час, що охоплює трансляцію, та додатковий час перед цим для виконання перевірок сигналу) оператори запускають конвеєр кодування через диспетчерську однією дією, яка координує ресурси в обох

регіонах. Інформаційна панель заповнюється прямими трансляціями від кодерів, що вносять свій вклад у місце проведення, та хмарних кодерів у кожному регіоні. Оператори можуть відстежувати критичні показники, а інтерфейс забезпечує швидкий доступ до ручного керування резервним мовленням – якщо в одному регіоні виникають проблеми, оператори можуть перенаправити трафік на справний конвеєр протягом кількох секунд.

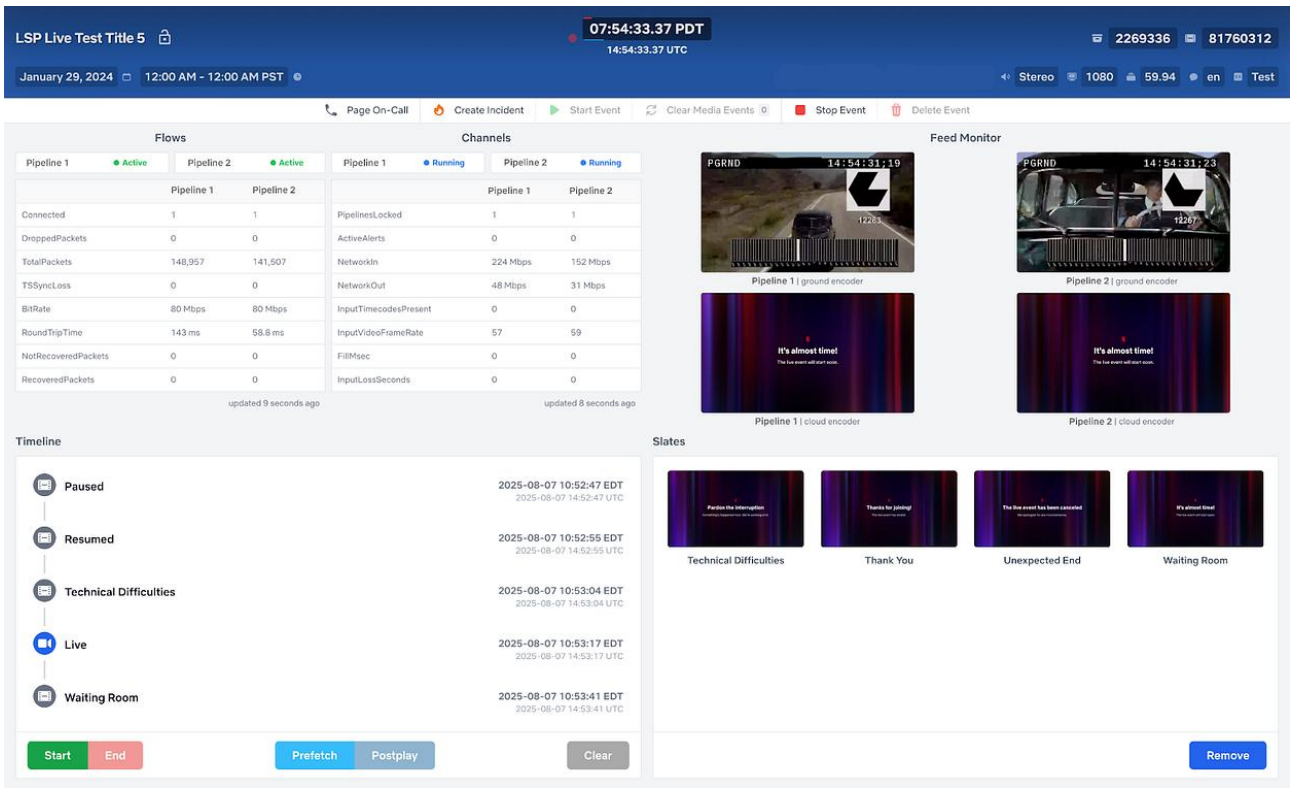

Рисунок 2.4 – Інтерфейс управління системами прямої трансляції

"Кімната керування" надає операторам інтерфейс для керування станом трансляції прямого ефіру на платформі протягом усього її життєвого циклу, від початку до кінця (або переходу до іншого шоу). Це гарантує, що досвід учасників відповідає фактичним часовим шкалам виробництва, адаптуючись до динамічного характеру подій у прямому ефірі.

Аналогічно, після завершення трансляції оператори повинні коректно вивести глядачів з потоку. Оскільки події в прямому ефірі не мають заздалегідь визначених кінцевих точок, це вимагає ручного втручання. Оператор ініціює кінцеву послідовність через диспетчерську, яка сигналізує клієнтам про вихід з

плеєра та точно позначає кінець вікна трансляції для подальшого перегляду на вимогу.

Хоча служба оркестрації автоматично обробляє складну конфігурацію та налаштування, "Кімната управління" надає операторам прозорість та контроль, необхідні для того, щоб кожна пряма трансляція забезпечувала якість, яку очікують споживачі контенту.

3 ОПИС ПРОЄКТУ ПРОПОНОВАНОЇ СИСТЕМИ

3.1 Обмеження для прямої трансляції та їх оптимізація

З огляду на мільйони пристроїв, підключених до Інтернету, та розклади подій у реальному часі, що можуть змінюватися, завдання побудови системи прямої трансляції полягає в тому, щоб забезпечити ідеальну синхронізацію всіх учасників. Щоби вирішити цю проблему, потрібно створити систему, яка не просто реагує, а адаптується, динамічно оновлюючи рекомендації в міру розвитку події. Тому є необхідність збалансувати три обмеження (див. рис. 3.1):

- Час: тривалість, необхідна для координації оновлення.
- Пропускна здатність запитів: здатність хмарних сервісів обробляти запити.
- Обчисліть кардинальність: різноманітність запитів, необхідних для обслуговування унікального оновлення.

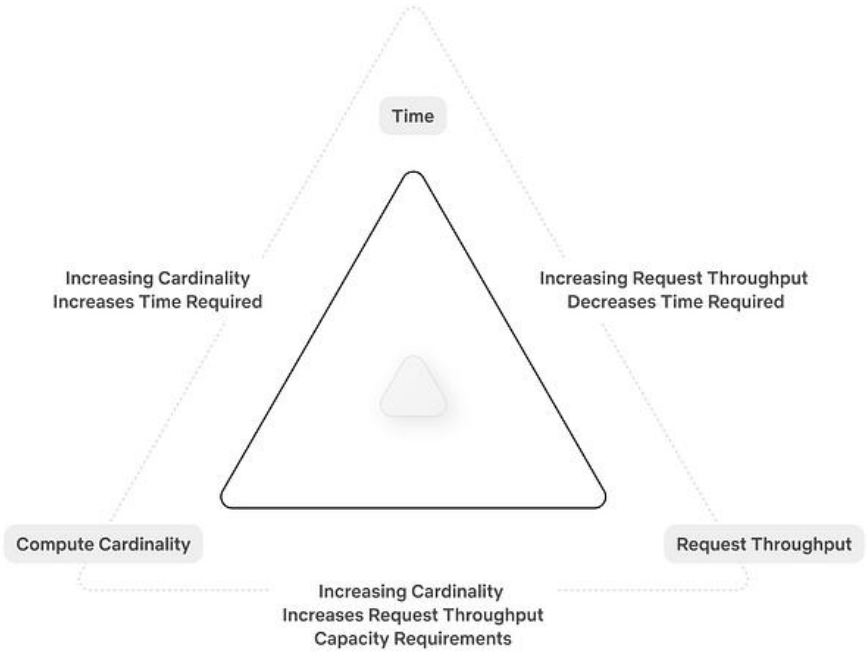

Рисунок 3.1 – Візуалізація обмежень для оновлень у режимі реального часу

Цю проблему оптимізації обмежень можна вирішити, розділивши рекомендації в реальному часі на дві фази: попередню вибірку та трансляцію в реальному часі. По-перше, слід заздалегідь вибрати необхідні дані, розподіляючи навантаження на довший період, щоб уникнути піків трафіку. Коли подія в прямому ефірі починається або закінчується, транслюється повідомлення з низькою кардинальністю на всі підключені пристрої, спонукаючи їх використовувати попередньо отримані дані локально. Час трансляції також адаптується, коли час події зміщується, щоб зберегти точність під час створення події в прямому ефірі. Поєднуючи ці дві фази, можна синхронізувати пристрої споживачів контенту та вирішити проблему "натовпу". Щоб максимізувати охоплення пристроїв, особливо для тих, хто має нестабільні мережі, застосовуються трансляції «принаймні один раз», щоб гарантувати, що кожен пристрій отримує останні оновлення та може надолужити будь-які раніше пропущені трансляції, щойно вони знову підключаться до мережі.

Перший етап оптимізує пропускну здатність запитів та кардинальність обчислень шляхом попереднього вибору рекомендацій, відображених метаданих заголовка та обкладинок для події в реальному часі. Оскільки учасники природним чином переглядають свої пристрої перед подією, ці дані попередньо заповнюються та зберігаються локально в кеші пристроїв, очікуючи на спрацьовування сповіщення для миттєвого надання рекомендацій. Розподіляючи ці запити природним чином протягом часу до події, можна усунути будь-які пов'язані з цим піки трафіку та уникнути необхідності значного масштабування системи в режимі реального часу (див. рис. 3.2).

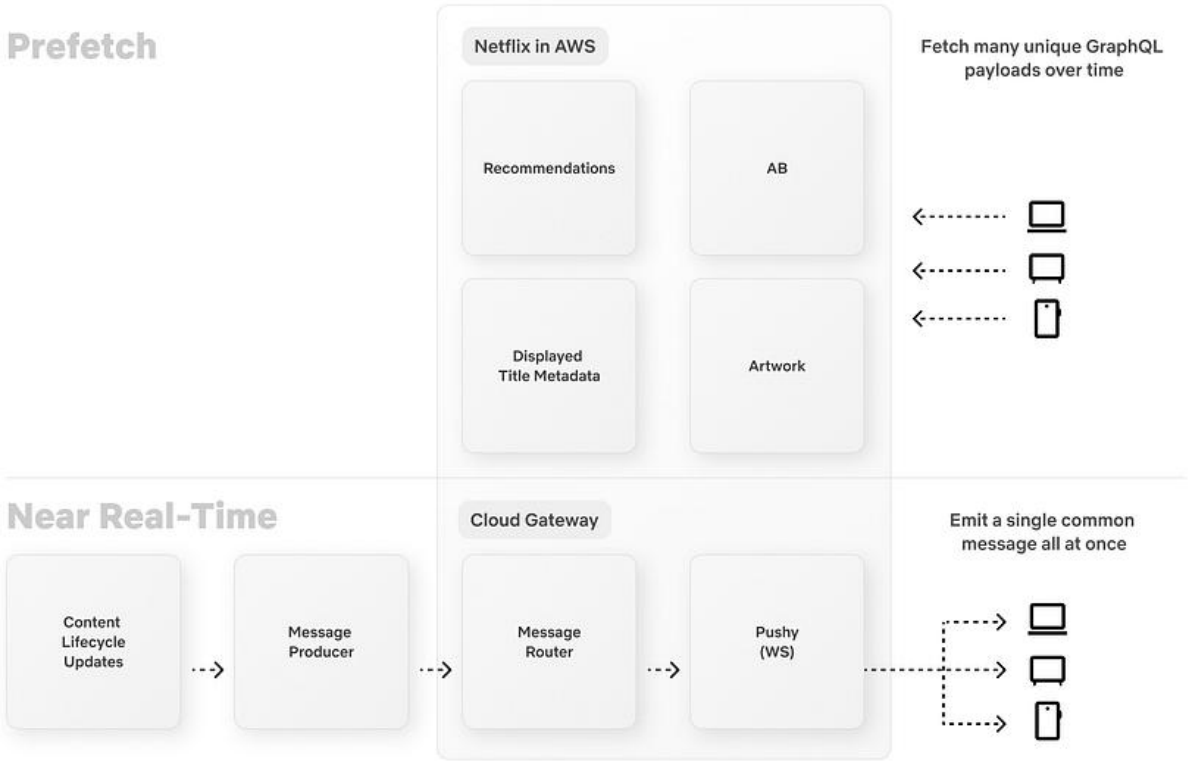

Рисунок 3.2 – Поетапний підхід, що згладжує запити трафіку з часом за допомогою широкомовної розсилки в режимі реального часу з низькою кардинальністю

Другий етап оптимізує пропускну здатність запитів та час оновлення. пристрої, транслюючи повідомлення в режимі реального часу з низькою кардинальністю на всі підключені пристрої в критичні моменти життєвого циклу події. Кожне корисне навантаження широкомовлення містить ключ стану та позначку часу. Ключ стану вказує на поточну стадію події, дозволяючи пристроям використовувати свої попередньо отримані дані для оновлення кешованих відповідей локально без додаткових запитів до сервера. Позначка часу гарантує, що якщо пристрій пропустить трансляцію через проблеми з мережею, він зможе надолужити пропущені оновлення після повторного підключення. Цей механізм гарантує, що пристрої отримають оновлення принаймні один раз, значно підвищуючи надійність доставки навіть у нестабільних мережах (див. рис. 3.3).

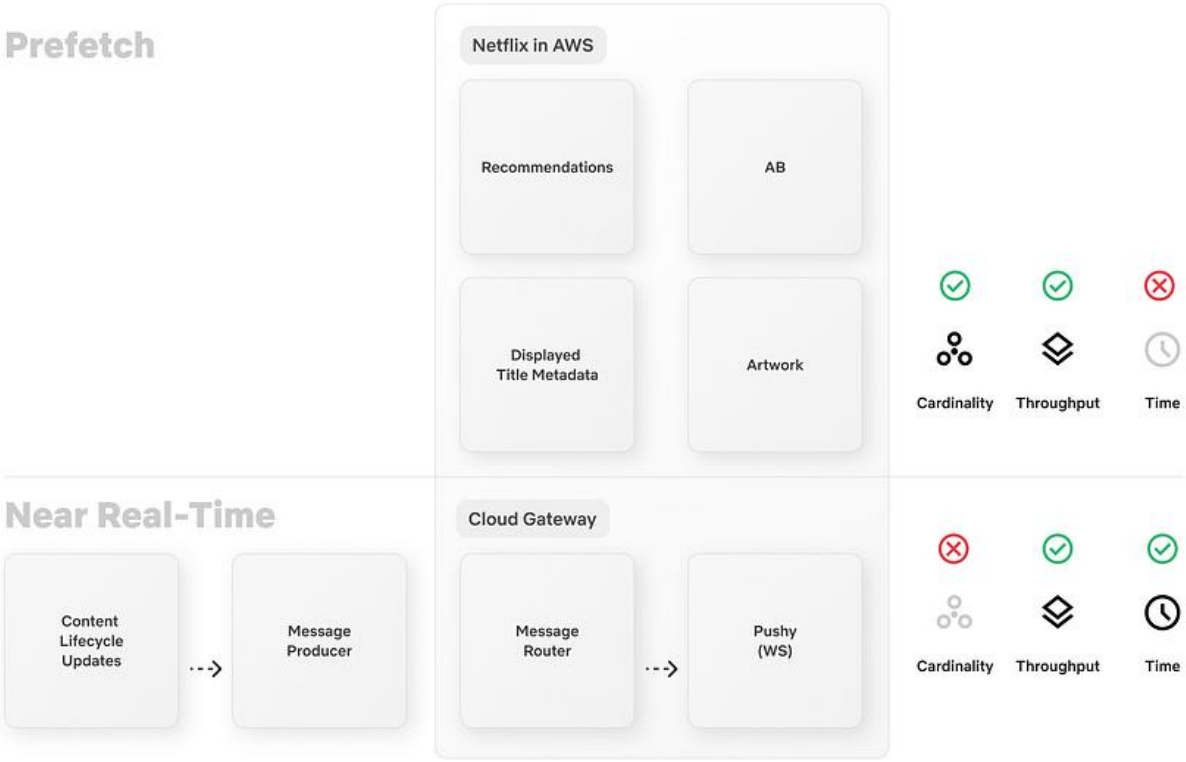

Рисунок 3.3 – Поетапний підхід оптимізації обмежень

Під час пікового навантаження можна успішно доставляти оновлення на кількох етапах онлайн-трансляцій на десятках мільйонів пристроїв менш ніж за хвилину.

3.2 Детальний аналіз роботи системи

Маючи на увазі загальну картину, розглянемо, як ці частини взаємодіють на практиці.

На діаграмі нижче (див. рис. 3.4) мікросервіс Message Producer централізує всю бізнес-логіку. Він постійно відстежує події в реальному часі на наявність змін у налаштуваннях та часі. Коли він виявляє оновлення, він планує розсилку широкомовних повідомлень саме в потрібний момент. Message Producer також стандартизує зв'язок, надаючи стислу схему GraphQL як для запитів пристроїв, так і для широкомовних корисних даних.

Замість надсилання широкомовних повідомлень безпосередньо на пристрої через WebSocket, Message Producer передає їх маршрутизатору повідомлень. Маршрутизатор повідомлень є частиною надійної дворівневої архітектури pub/sub, побудованої на перевірених технологіях, таких як Pushy (проксі-сервер WebSocket), Apache Kafka та сховище ключів-значень KV. Маршрутизатор повідомлень відстежує підписки на рівні деталізації вузла Pushy, тоді як вузли Pushy зіставляють підписки з окремими з'єднаннями, створюючи низьку затримку, яка мінімізує вимоги до обчислень та пропускної здатності.

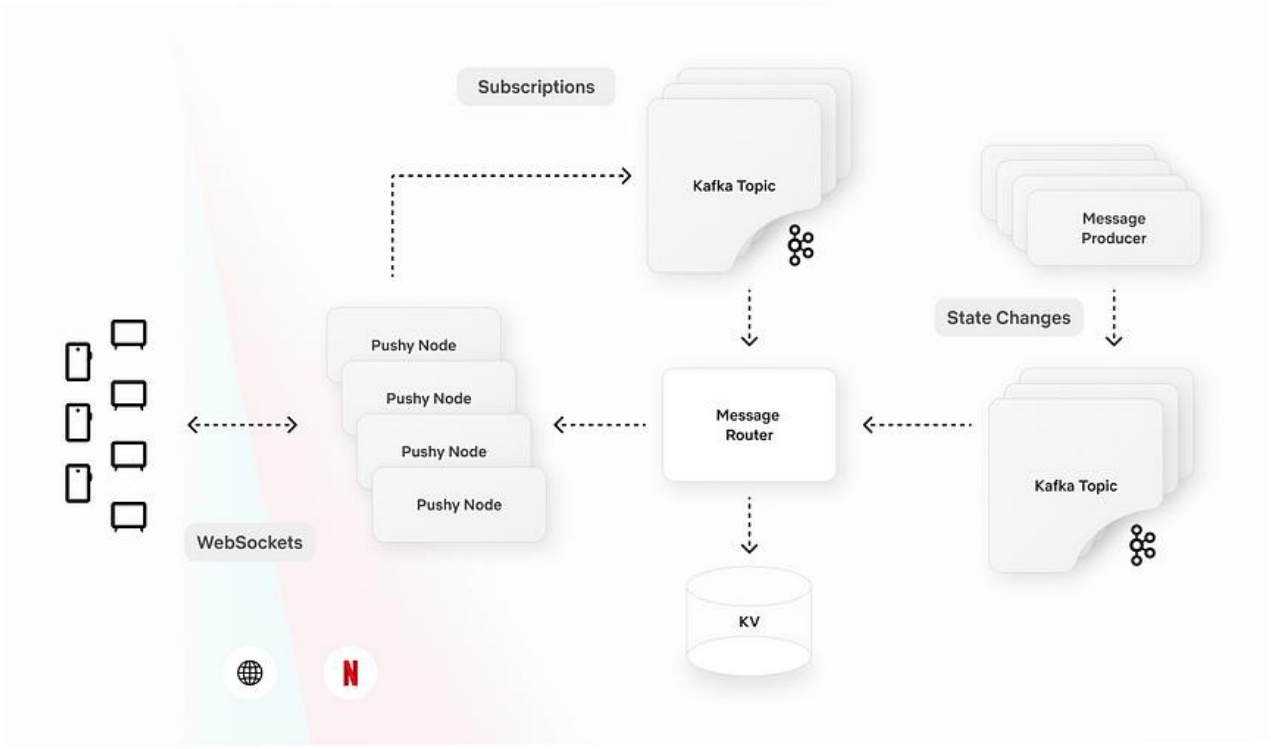

Рисунок 3.4 – Архітектура мікросервісу Message Producer

Пристрої взаємодіють з GraphQL (Служба графів доменів – DGS). Ці схеми пропонують кілька інтерфейсів запитів для попередньої вибірки, що дозволяє пристроям адаптувати свої запити до конкретного кінцевого пристрою згідно вподобань його користувача. Кожна відповідь дотримується узгодженого API, який розв'язується як карта ключів етапів, що забезпечує швидкий пошук і не додає бізнес-логіки до пристрою. Схема широкомовлення визначає параметри

підключення WebSocket, поточний етап події та позначку часу останнього широкомовного повідомлення. Коли пристрій отримує широкомовне повідомлення, він вставляє корисне навантаження безпосередньо в свій кеш, запускаючи негайне оновлення та повторний рендеринг інтерфейсу.

Окрім створення нової технології для підтримки рекомендацій у режимі реального часу, також проведено оцінювання існуючих системи на наявність потенційних гарячих точок трафіку. Використовуючи прогнози трафіку з високим рівнем трафіку для прямих трансляцій, було створено синтетичний трафік для імітації сценаріїв та проведено спостереження, як онлайн-сервіси справляються з цими сплесками. У ході цього процесу було виявлено кілька спільних закономірностей

1. Порушення синхронізації кешу.

Симуляції події прямої трансляції показали, що хоча запропонований підхід зменшував безпосередні ризики лавиноподібного наростання запитів до серверів, спричинені трафіком користувачів під час подій, події в реальному часі призвели до неочікуваних сплесків у системах за кілька годин до та після самих подій. Різке зростання кількості учасників, які приєднувалися якраз вчасно до цих подій, призвело до концентрованого закінчення терміну дії кешу та переобчислень, що створювало сплески трафіку далеко за межами вікна події, яких не очікували. Це не було проблемою для VOD-контенту, оскільки моделі трафіку учасників набагато плавніші. Виявляється, що фіксовані значення TTL призводили до одночасного закінчення терміну дії кешу та сплесків трафіку оновлення. Щоб вирішити цю проблему, було додано випадковий зсув до закінчення терміну дії кешу сервера та клієнта, щоб розподілити оновлення та згладити сплески трафіку.

2. Адаптивне визначення пріоритетів трафіку.

Хоча сервіси вже використовують пріоритизацію та розподіл трафіку на основі таких факторів, як тип запиту та тип пристрою, події в реальному часі створили особливу проблему. Ці події генерували короткі сплески трафіку, які були надзвичайно різкими та створювали значне навантаження на системи.

Щоб вирішити цю проблему, було вдосконалили стратегії шардування трафіку, використовуючи сигнали на основі подій. Це дозволило направляти трафік подій прямих трансляцій до виділених кластерів з більш агресивними політиками масштабування. Також було добавлено набір правил динамічного пріоритетування трафіку, який активується щоразу, коли спостерігається висока кількість запитів за секунду (RPS), щоб забезпечити безперебійну роботу наших систем зі сплеском навантаження (див. рис. 3.5).

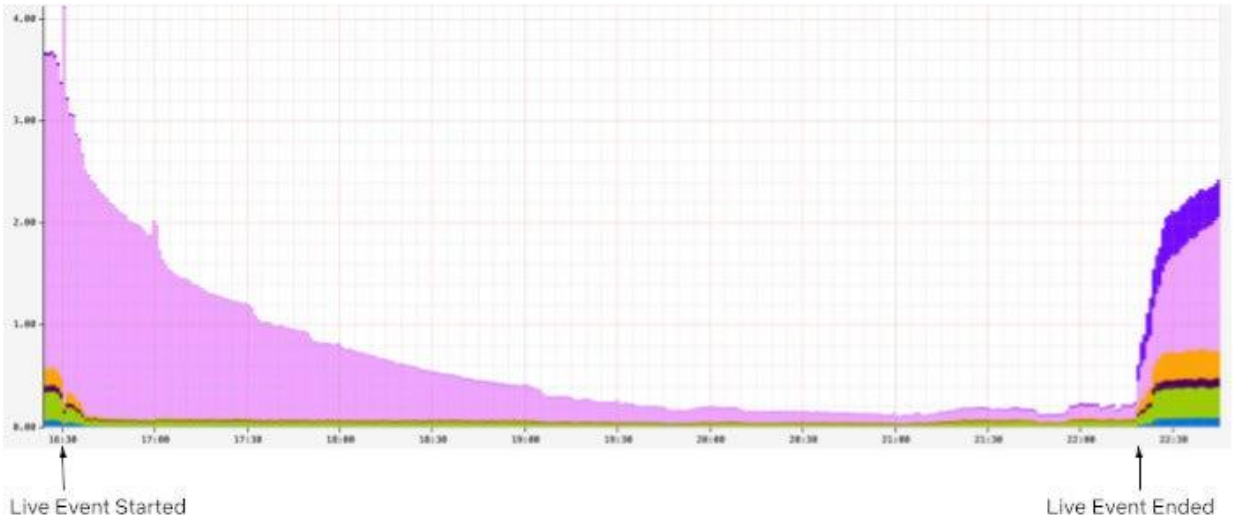

Рисунок 3.5 – Знімок падіння обсягу некритичного трафіку (у %) для сервісу, орієнтованого на учасників, під час прямої трансляції досягнуто шляхом агресивного зниження пріоритетів

Під час цих піків примусово знижується пріоритет некритичних оновлень, ініційованих сервером, щоб системи могли виділити ресурси на найбільш чутливі до часу обчислення. Такий підхід забезпечує безперебійну роботу та надійність, коли навантаження найвище.

3.3 Приклад простої стрімінгової системи на основі Apache Kafka

В якості демонстрації наведемо простий проєкт для трансляції відеопотоку, який дає можливість проілюструвати роботу спрощеного клієнт-серверного застосунку для розповсюдження відеоконтенту. Відеопотік з цього

застосунку можна безпосередньо демонструвати на кінцевих пристроях чи використовувати для подальшого опрацювання, такого, як, наприклад, класифікація об'єктів.

Повноцінний демонстраційний додаток з використанням Apache Kafka та зі створенням продюсера і споживача для потокової передачі відео. Для ілюстрації вибрано зразок відео з джерела RTSP, використовуючи URL-потік через продюсера в мережі Docker. Продюсер створений на мові Go для надсилання кадрів до теми (топіка) Kafka.

Архітектура створеного додатка показана на рисунку 3.6.

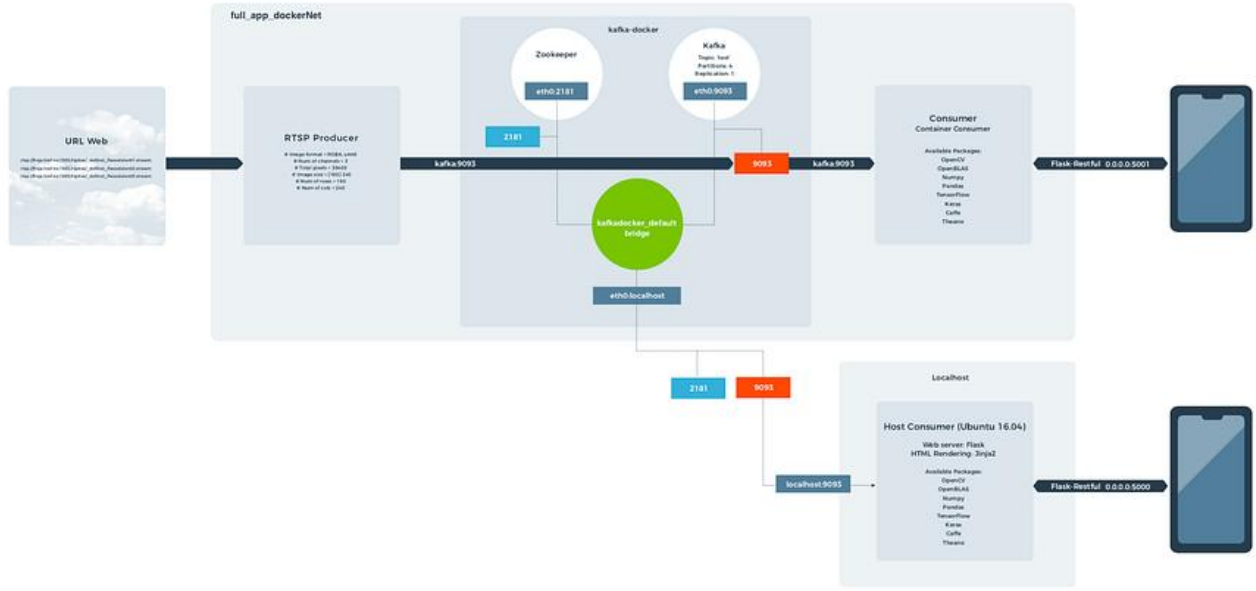

Рисунок 3.6 – Архітектура демонстраційного додатку

У цьому додатку у є два споживачі. Один споживач працює всередині мережі контейнерів, взаємодіючи з каналом Kafka через порт kafka:9093. Інший призначений для запуску з локального хоста, який отримує доступ до каналу Kafka через порт localhost:9092. В обох випадках можна створити образ Docker та локальне середовище, що містить ті самі пакети, здебільшого включаючи популярний фреймворк Computer Vision (OpenCV) та популярні фреймворки ML (TF, Keras, Theano та Caffe). Очевидно, що не потрібно використовувати всі

фреймворки, але це гарний приклад, який можна використати для майбутніх проєктів.

Після запуску системи в докер-контейнерах можна побачити споживачів, що працюють на веб-сервері на базі Flask.

До споживача в мережі Docker можна отримати доступ: http://0.0.0.0:5001

До споживача, що працює з локального хоста, можна отримати доступ: http://0.0.0.0:5000

Якщо є потреба змінити споживача, то треба перейти до папки споживача. Там знайти файл main.py. Далі можна додати потрібні алгоритми машинного навчання, де буде прямий доступ до пікселів зображення у функції get_stream():

```
feed = msg.value.get("pix")
```

Або якщо є потреба отримати дані в байтах:

```
b = bytes(feed, 'utf-8')
```

Список модулів для опрацювання відео подано у таблиці 3.1.

Таблиця 3.1 – Список включених модулів

Modules	**Versioning**	**Modules**	**Versioning**
darknet	latest	opencv	4.0.1
python	3.6	sonnet	latest
torch	latest	caffe	latest
chainer	latest	cntk	latest
mxnet	latest	flask	latest
omx	latest	numpy	latest
pytorch	latest	dlib	latest
tensorflow	latest	facial-recognition	latest
theano	latest	Jinja2	latest

Цей проєкт слугує легкою демонстрацією для конвеєра потокового відео. Вам не слід використовувати цей проєкт у робочому середовищі, але це гарний спосіб розпочати роботу з Kafka, Flask та бібліотеками машинного навчання, усі з яких використовують Python.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Методи та засоби психофізіологічного розвантаження як допоміжний процес в розробці ПЗ

Для галузі ІТ інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [13].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [17]. Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

1) надмірне інформаційне навантаження, суть якого полягає у тому, що кількість корисної інформації, яка надходить до мережі, перевищує психофізіологічні можливості її сприйняття людиною;

2) велика кількість інформації, яка сприймається, але не є корисною для фахівців в даний момент;

3) інформаційний голод, причиною якого є саме надлишок інформації, викликаний інформаційним перенавантаженням;

4) "інформоманія" як хвороба людини, яка робить останню знеособленою, залежною від перебування в інформаційному просторі і роботи з комп’ютером і чому вона віддає перевагу, уникаючи "живого" спілкування з людьми;

5) поява "кіберспільнот", що за своїми соціокультурними характеристиками набагато ближчі до представників інших культур у

глобальному інформаційному просторі, ніж до своєї етнонаціональної спільноти чи решти населення, не охопленого Інтернетом;

6) індивідуалізм і дегуманізація способу життя "мешканців" Інтернету – відсутність готовності ділитися своїми знаннями.

Слід розуміти, що комп'ютерні технології істотно впливають на життєдіяльність людини, припускаючи глобалізацію і технократизацію суспільства. Але в ще більшій мірі цей вплив поширюється безпосередньо на центральну нервову систему, яка звикає працювати в дуже інтенсивному режимі багатозадачності, де вже переважають не тривалі логічні роздуми, а інтуїтивно-реактивні ланцюжки розумових формулювань у зв'язку з величезним обсягом оброблюваної щодня інформації, кількість якої зростає за експоненціальною швидкістю. Виникає припущення, що саме збільшення обсягу інформації та прискорення її обробки людиною може згубно вплинути на розвиток розумових здібностей людини.

У [13] наведено перелік протипоказань з боку органів зору та загальних (соматичних) протипоказань, які забороняють роботу на ЕОМ, а також комплекс вправ для поліпшення здоров’я і підвищення працездатності.

Таким чином, за умови високого рівня робіт з ЕОМ рекомендується психофізіологічне розвантаження у спеціально обладнаних приміщеннях (кімнати психофізіологічного розвантаження) під час регламентованих перерв або в кінці робочого дня.

При проведенні сеансів психофізіологічного розвантаження рекомендується використовувати деякі елементи методу аутогенного тренування, який ґрунтується на свідомому застосуванні комплексу взаємопов’язаних прийомів психічної саморегуляції й виконанні нескладних фізичних вправ зі словесним самонавіюванням.

У рекомендованому сеансі, який має проводитися у кімнаті психофізіологічного розвантаження з відповідним інтер'єром та кольоровим оформленням, виділяють такі три періоди, або фази.

Перший – абстрагування працівників від виробничої обстановки – відповідає фазі залишкового збудження. Звучить повільна мелодійна музика, пташиний спів. Обравши зручну позу, працівники адаптуються і психологічно готуються до наступних періодів.

Другий – заспокоєння – відповідає фазі відновлювального гальмування. Пропонується показ фото слайдів із зображеннями квітучого лугу, березового гаю, гладенької поверхні ставка тощо. Через навушники транслюється спокійна музика.

Як функціональне освітлення застосовують зелене світло. Яскравість світла має поступово знижуватися впродовж періоду заспокоєння, а наприкінці його світло вимикається зовсім на 1– 2 хв. Екран теж гасне.

Третій – активізація – відповідає фазі підвищеної збудженості.

На початку періоду світло вимкнене, через певний час на екрані з'являється червона пляма, розміри й яскравість якої поступово збільшуються.

Наприкінці періоду звучить бадьора музика. Вимовляються тричі мобілізуючі формули аутогенного тренування, яким мають передувати глибокий вдих та довгий глибокий видих.

Після сеансів психофізіологічного розвантаження у працівників зменшується відчуття втоми, з'являється бадьорість, хороший настрій. Загальний стан відчутно поліпшується.

4.2 Попередження аварій на виробництвах із застосуванням хлору

Хлор є частиною таблиці хімічних елементів і розташовується в ній під номером 17. У природі він зустрічається виключно у формі газу. Найчастіше він має специфічний зелений з жовтим переливом колір. Цей елемент важчий за повітря в 2,5 рази, тому накопичується в підвалах будинків, а на пересіченій місцевості в ярах і низинах. У воді ж хлор розчиняється без сліду і його наявність помітно тільки при великій концентрації (за рахунок специфічного запаху) [14].

В організмі людини в середньому міститься 95 г хлору. За добу людина споживає 5-10 г хлору (кухонна сіль). Він потрібен для вироблення в шлунку соляної кислоти, яка сприяє травленню і знищенню хвороботворних бактерій. Добова потреба хлору для людини становить 800 мг.

Хлор широко застосовується на виробництві, на його основі виготовляють отрутохімікати, розчинники, засоби для дезінфекції та миття, медикаменти. Хлор використовується в кольоровій металургії, у виготовленні пластмас тощо. Також хлор з успіхом застосовується і в побуті для очищення, відбілювання, прання. Завдяки незначним витратам і досить високої ефективності дезінфекції, хлор активно використовується для очищення і знезараження води в плавальних басейнах і питної водопровідної води.

Отруєння хлором можливе в разі:

– перевищення максимально допустимих концентрацій хлору для знезараження води в трубопроводі (сильний запах хлору);

– наявність хлору у великій кількості у воді басейну і часте купання в ньому;

– відбілювання і прання в закритому не провітрюваному приміщенні;

– аварії на підприємстві;

– використання хлору в якості зброї масового ураження.

В організм хлор потрапляє через слизові оболонки дихальної і травної систем, шкіру.

Ознаки отруєння хлором. До перших ознаках отруєння хлором відносяться:

– дискомфорт і подразнення слизової дихальних шляхів;

– підвищене слиновиділення і спазм голосових зв’язок;

– кашель і утруднене дихання;

– відчуття різі та печіння в очах, сльозотеча;

– нудота і гіркота у роті;

– головні болі і можливі судоми.

При попаданні на шкірний покрив або слизові спостерігається значний свербіж і гіперемія (почервоніння), вірогідні підшкірні крововиливи без пошкодження цілісності шкіри.

Тяжкість патологічного процесу та симптоми отруєння хлором знаходяться в прямій залежності від дози отруйної речовини (хлору) і тривалості його дії.

До прибуття медиків слід надати домедичну допомогу потерпілому:

– усунути джерело надходження отрути в організм – вивести або винести потерпілого поза зону дії отруйної речовини. При цьому необхідно пам’ятати про безпеку рятувальника – застосування марлевої маски або респіратора.

– забезпечити доступ чистого повітря;

– зняти забруднений одяг і теплою (не гарячою) водою промити контактуючі ділянки шкіри.

– у разі перорального надходження (проковтування) хлоровмісних рідин, потрібно промити шлунок. Промивати краще через зонд, або можна викликати блювання після рясного пиття.

– у разі пошкодження очей, промивання великою кількістю води або слабким розчином соди для зняття подразнення;

– полоскання ротової порожнини та носа содовими розчинами для мінімізації ушкодження слизових оболонок, застосування інгаляцій з додаванням соди для полегшення кашлю.

До профілактичних заходів отруєння хлором належать:

– забезпечення належних умов праці відповідно до санітарно-технічних вимог (вентиляція, провітрювання, справне обладнання);

– використання індивідуальних засобів захисту при роботі з хімікатами на виробництві;

– регулярні перевірки концентрацій хлору в повітрі робочої зони;

– проведення профілактичних медичних оглядів для виявлення схильності (доклінічних форм) і хронічних захворювань;

– дотримання вимог безпеки у використанні хлоровмісних рідин в побуті.

Суб'єкт господарської діяльності зобов'язаний забезпечити працівників хлорних об'єктів спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту відповідно до Положення про порядок забезпечення працівників спеціальним одягом, спеціальним взуттям та іншими засобами індивідуального захисту:

а) для захисту органів дихання – фільтруючими протигазами, ізолюючими дихальними апаратами та ізолюючими костюмами;

б) для захисту очей – захисними окулярами;

в) для захисту шкіри від їдких речовин – гумовими або прогумованими рукавицями, гумовими чоботами або шкіряними черевиками, сукняними костюмами.

Враховуючи значний ризик і широке застосування хлору, тяжкість ураження і високу можливість летального наслідку, у кожного повинен бути сформований алгоритм дій і чітка позиція – попередити отруєння легше і доцільніше, ніж лікувати і боротися з його наслідками.

ВИСНОВКИ

Якщо підвести підсумки виконаного дослідження, то можна сформулювати такі висновки.

Спроби перенаправити трафік на інші некритичні сервіси призводять до неочікуваних моделей сплесків трафіку в інших місцях. Аналогічно високий обсяг трафіку від популярних подій спричиняє надмірне необов'язкове ведення журналу та створює непотрібне навантаження на канали обробки.

Якщо перейти до більш детального формулювання висновків, то можемо відмітити наступне.

У першому розділі було здійснено огляд платформи Netflix та її архітектури рекомендаційної системи для трансляції прямих ефірів в масштабі реального часу. Особливу увагу приділено технічним аспектам прямих трансляцій, які потребують масштабованої інфраструктури, здатної забезпечити високу якість сигналу та стабільність відтворення на мільйонах пристроїв. Використання CDN, хмарних сервісів та спеціалізованих алгоритмів кодування дозволяє інтегрувати Live-формат у загальний користувацький досвід без втрати якості.

Другий розділ присвячено аналізу предметної області та сучасних підходів до побудови рекомендаційних систем. Розглянуто застосування мовних моделей та алгоритмів машинного навчання для формування релевантних рекомендацій у реальному часі. Визначено основні виклики, серед яких — масштабування моделей для роботи з великими масивами даних, забезпечення низької затримки при обробці запитів та інтеграція з системами відтворення контенту. Показано, що використання архітектур на основі Kubernetes або власних рішень дозволяє ефективно управляти ресурсами та забезпечувати гнучкість системи. Водночас підкреслено важливість балансування між якістю рекомендацій та продуктивністю інфраструктури.

У третьому розділі описано проєкт запропонованої системи рекомендацій для онлайн-стрімінгових платформ. Сформульовано підхід до ранжування

контенту на основі рейтингу та взаємодії користувачів, що дозволяє підвищити точність рекомендацій. Розглянуто джерела даних, які включають як явні оцінки, так і неявні сигнали (перегляди, кліки, час відтворення). Виконано дослідження даних та сформовано гіпотези щодо поведінки користувачів, що стало основою для побудови моделі. Проведено розробку та тестування системи, результати яких підтвердили можливість динамічного оновлення рекомендацій у режимі реального часу без перевантаження серверів. Практичне значення полягає у створенні архітектури, здатної масштабуватися до сотень тисяч одночасних користувачів, забезпечуючи персоналізований досвід та високу якість відеосигналу.

Загалом, результати дослідження свідчать про те, що інтеграція сучасних алгоритмів машинного навчання з масштабованою інфраструктурою стрімінгових платформ дозволяє створити ефективну рекомендаційну систему для прямих трансляцій. Запропоновані рішення можуть бути використані у сервісах, що прагнуть забезпечити користувачам персоналізований доступ до контенту в реальному часі. Це відкриває перспективи для подальшого розвитку технологій онлайн-трансляцій та підвищення конкурентоспроможності стрімінгових платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Behnam N. Using Traffic Modeling to Load-Balance Netflix Traffic at Global Scale. InfoQ. URL: https://www.infoq.com/presentations/load-balancing-netflix/ (date of access: 16.10.2025).

2. Vora M., Berglund A., Sadafal V. Distributing Content to Open Connect. Distributing Content to Open Connect. URL: https://netflixtechblog.com/distributing-content-to-open-connect-3e3e391d4dc9 (date of access: 11.09.2025).

3. Keck C. A look under the hood of the most successful streaming service on the planet. The Verge. URL: https://www.theverge.com/22787426/netflix-cdn-open-connect (date of access: 16.11.2025).

4. Lives Ceptro - Home. URL: https://intrarede.nic.br/files/apresentacao/arquivo/850/Live%20Intra%20REDE%20Nic.br%202020.pdf (дата звернення: 10.12.2025).

5. Vora M., Berglund A., Sadafal V. Introducing Netflix's Key-Value Data Abstraction Layer. Medium.com. URL: https://netflixtechblog.com/introducing-netflixs-key-value-data-abstraction-layer-1ea8a0a11b30 (date of access: 16.10.2025).

6. Шумова, Л. О., Рязанцев, О. І., & Покришка, С. А. (2023). Моделі машинного навчання для формування рекомендацій. Вісник Східноукраїнського національного університету імені Володимира Даля, (2 (278)), 96-105.

7. ШЕВЧЕНКО, С., ЖДАНОВА, Ю., & ДАНИЛЮК, О. (2024). АНАЛІЗ ТА ДОСЛІДЖЕННЯ ХАРАКТЕРИСТИК АЛГОРИТМІВ У РЕКОМЕНДАЦІЙНИХ СИСТЕМАХ. Вісник Херсонського національного технічного університету, (4 (91)), 367-376.

8. Пасічник, В. В., Юнчик, В. Л., Кунанець, Н. Е., & Федонюк, А. А. (2022). Використання нечіткої логіки у процесі експертного оцінювання електронних навчальних ресурсів. Scientific Bulletin of UNFU, 32(4), 66-76.

9. Pankiv, Y., Kunanets, N., Artemenko, O., Veretennikova, N., & Nebesnyi, R. (2021, September). Project of an intelligent recommender system for parking

vehicles in smart cities. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 2, pp. 419-422). IEEE.

10. Matsyuk, O., Nazaruk, M., Turbal, Y., Veretennikova, N., & Nebesnyi, R. (2018, September). Information analysis of procedures for choosing a future specialty. In Conference on Computer Science and Information Technologies (pp. 364-375). Cham: Springer International Publishing.

11. Небесний, Р. М. (2023). Рекомендаційна система формування команд виконавців з відповідними фаховими компетентностями (Doctoral dissertation, Тернопільський національний технічний університет імені Івана Пулюя).

12. Ржеуський, А., Кунанець, Н., & Стахів, М. (2018). Рекомендаційна система інформаційного обслуговування користувачів бібліотек. У Матеріали V науково-технічної конференції "Інформаційні моделі, системи та технології" (с. 37). Тернопіль: ТНТУ.

13. Голінько В. І. Охорона праці в галузі інформаційних технологій: навч. посіб. / В. І. Голінько, М. Ю. Іконніков, Я. Я. Лебедєв; М-во освіти і науки України, Держ. вищий навч. закл. "Нац. гірн. ун-т". - Дніпропетровськ: НГУ, 2015. - 246 с.

14. Гандзюк М.П. Основи охорони праці: Підручник. 4-е вид./Гандзюк М.П., Желібо Є.П., Халімовський М.О. - Київ: Каревела, 2008. – 384с.

15. Техноекологія та цивільна безпека. Частина «Цивільна безпека»: Навчальний посібник; укл.: Стручок В. С. Тернопіль: ФОП Паляниця В.А., 2022. 150 с.

16. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.

17. Умови праці працівників, які використовують у роботі персональні комп'ютери. Zolochiv.Net. URL: https://zolochiv.net/umovy-pratsi-pratsivnykiv-iaki-vykorystovuiut-u-roboti-personal-ni-komp-iutery/ (дата звернення: 25.10.2024).

ДОДАТКИ

Додаток А

Тези доповіді

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя
Маріборский університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Краківський економічний університет (Польща)
Вроцлавський економічний університет (Польща)
Університет «Опольська Політехніка» (Польща)
Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Вінницький національний аграрний університет
Львівський національний університет ім. І. Франка
Головне управління Пенсійного фонду в Тернопільській області
Наукове товариство ім. Шевченка
Тернопільський обласний комунальний інститут післядипломної педагогічної освіти
Сумський державний педагогічний університет
Запорізький національний університет

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

XIV Міжнародної науково-технічної конференції молодих учених та студентів

11-12 грудня 2025 року

УКРАЇНА
ТЕРНОПІЛЬ – 2025

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.

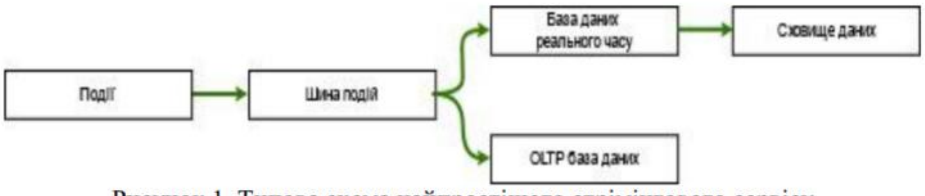

Рисунок 1. Типова схема найпростішого стрімінгового сервісу

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.

Додаток Б

Програмний код

Продюсер:

```
package main

import (
     "encoding/json"
     "fmt"
     "image"
     "io"
     "log"
     "os"
     "time"
     "github.com/Shopify/sarama"
     "github.com/adaickalavan/kafkapc"
     "gocv.io/x/gocv"
)

//Hooks that may be overridden for testing
var inputReader io.Reader = os.Stdin
var outputWriter io.Writer = os.Stdout

// Instantiate a producer
var producer2 sarama.AsyncProducer

// Instantiate the Kafka topic
var ntopic string = "camera2"

// Instantiate the RSTP Link
var rstp_link =
"rtsp://freja.hiof.no:1935/rtplive/_definst_/hessdalen02.stream"

func main() {
     //Sarama logger
     sarama.Logger = log.New(outputWriter, "[saramaLog]",
log.Ltime)

     //Create a Kafka producer
     var brokers = []string{"kafka:9093"}
     var err error
     producer2, err = kafkapc.CreateKafkaProducer(brokers)
     if err != nil {
          panic("Failed to connect to Kafka. Error: " +
err.Error())
     }
     //Close producer to flush(i.e., push) all batched messages
into Kafka queue
     defer func() { producer2.Close() }()

     // Capture video
```

```
	webcam, err := gocv.OpenVideoCapture(rstp_link)
	if err != nil {
		panic("Error in opening webcam: " + err.Error())
	}

	// Stream images from RTSP to Kafka message queue
	frame := gocv.NewMat()
	for {
		if !webcam.Read(&frame) {
			continue
		}

		resultImage := gocv.NewMatWithSize(512, 288,
gocv.MatTypeCV8U)
		gocv.Resize(frame, &resultImage,
image.Pt(resultImage.Rows(), resultImage.Cols()), 0, 0,
gocv.InterpolationCubic)

		// Type assert frame into RGBA image
		imgInterface, err := resultImage.ToImage()
		if err != nil {
			panic(err.Error())
		}
		img, ok := imgInterface.(*image.RGBA)
		if !ok {
			panic("Type assertion of pic (type image.Image
interface) to type image.RGBA failed")
		}

		//Form the struct to be sent to Kafka message queue
		doc := Result{
			Pix:      img.Pix,
			Channels: frame.Channels(),
			Rows:     frame.Rows(),
			Cols:     frame.Cols(),
		}

		//Prepare message to be sent to Kafka
		docBytes, err := json.Marshal(doc)
		if err != nil {
			log.Fatal("Json marshalling error. Error:",
err.Error())
		}
		msg := &sarama.ProducerMessage{
			Topic:     ntopic,
			Value:     sarama.ByteEncoder(docBytes),
			Timestamp: time.Now(),
		}
		//Send message into Kafka queue
		producer2.Input() <- msg

		//Print time of receiving each image to show the code is
running
```

```
          fmt.Fprintf(outputWriter, "Sending Camera 2 Bytes ----
>>>> %v\n", time.Now())
     }
}

//Result represents the Kafka queue message format
type Result struct {
     Pix      []byte `json:"pix"`
     Channels int    `json:"channels"`
     Rows     int    `json:"rows"`
     Cols     int    `json:"cols"`
}
```

Консюмер (споживач)

```
# Define Imports
import base64
from json import loads

import cv2
import numpy as np
from flask import Flask, Response, render_template
from kafka import KafkaConsumer

W = 512
H = 288

# Instantiate Kafka Consumers
consumer = KafkaConsumer(
    'camera2',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group-1',
    value_deserializer=lambda m: loads(m.decode('utf-8')),
    bootstrap_servers=['kafka:9093'])

consumer2 = KafkaConsumer(
    'test',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group-1',
    value_deserializer=lambda m: loads(m.decode('utf-8')),
    bootstrap_servers=['kafka:9093'])

# Instantiate Kafka Consumers
consumer3 = KafkaConsumer(
    'camera2',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group-1',
    value_deserializer=lambda m: loads(m.decode('utf-8')),
    bootstrap_servers=['kafka:9093'])
```

```
consumer4 = KafkaConsumer(
    'test',
    auto_offset_reset='earliest',
    enable_auto_commit=True,
    group_id='my-group-1',
    value_deserializer=lambda m: loads(m.decode('utf-8')),
    bootstrap_servers=['kafka:9093'])

# Set the App to be Flask based
app = Flask(__name__)


# Route Index Page
@app.route('/', methods=['GET'])
def Index():
    """
    Using Jinja we are rendering a template page
    where it will call the get_stream() function
    """
    return render_template('index.html')


@app.route('/video_feed')
def video_feed():
    return Response(get_stream(), mimetype='multipart/x-mixed-
replace; boundary=frame')


@app.route('/video_feed2')
def video_feed2():
    return Response(get_stream2(), mimetype='multipart/x-mixed-
replace; boundary=frame')


@app.route('/video_feed3')
def video_feed3():
    return Response(get_stream3(), mimetype='multipart/x-mixed-
replace; boundary=frame')


@app.route('/video_feed4')
def video_feed4():
    return Response(get_stream4(), mimetype='multipart/x-mixed-
replace; boundary=frame')


def get_stream():
    print('Listening to Feed 1...')
    count = 0
    for msg in consumer2:
        # Conversion: base-64 string --> array of bytes --> array
of integers
```

```
        val = msg.value
        base64string = val['pix']  # pix is base-64 encoded string
        byteArray = base64.b64decode(base64string)  # byteArray is
an array of bytes
        npArray = np.frombuffer(byteArray, np.uint8)  # npArray is
an array of integers

        # Reshape array into an RGB image matrix of shape
(channels, rows, cols)
        rows = val['rows']
        cols = val['cols']
        channels = val['channels']
        imgR = npArray[0::4].reshape((H, W))
        imgG = npArray[1::4].reshape((H, W))
        imgB = npArray[2::4].reshape((H, W))
        img = np.stack((imgR, imgG, imgB))
        img = np.moveaxis(img, 0, -1)
        success, a_numpy = cv2.imencode('.jpg', img)
        count += 1
        print('Feed 1: Displaying packet {}'.format(count))
        a = a_numpy.tostring()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpg\r\n\r\n' + a +
b'\r\n\r\n')


def get_stream2():
    print('Listening to Feed 2...')
    count = 0
    for msg in consumer:
        # Conversion: base-64 string --> array of bytes --> array
of integers
        val = msg.value
        base64string = val['pix']  # pix is base-64 encoded string
        byteArray = base64.b64decode(base64string)  # byteArray is
an array of bytes
        npArray = np.frombuffer(byteArray, np.uint8)  # npArray is
an array of integers

        # Reshape array into an RGB image matrix of shape
(channels, rows, cols)
        rows = val['rows']
        cols = val['cols']
        channels = val['channels']
        imgR = npArray[0::4].reshape((H, W))
        imgG = npArray[1::4].reshape((H, W))
        imgB = npArray[2::4].reshape((H, W))
        img = np.stack((imgR, imgG, imgB))
        img = np.moveaxis(img, 0, -1)
        success, a_numpy = cv2.imencode('.jpg', img)
        a = a_numpy.tostring()
        count += 1
        print('Feed 2: Displaying packet {}'.format(count))
```

```
        yield (b'--frame\r\n'
               b'Content-Type: image/jpg\r\n\r\n' + a +
b'\r\n\r\n')


def rgb2gray(rgb):
    return np.dot(rgb[..., :3], [0.2989, 0.5870, 0.1140])


def get_stream3():
    print('Listening to Feed 3...')
    count = 0
    for msg in consumer3:
        # Conversion: base-64 string --> array of bytes --> array
of integers
        val = msg.value
        base64string = val['pix']  # pix is base-64 encoded string
        byteArray = base64.b64decode(base64string)  # byteArray is
an array of bytes
        npArray = np.frombuffer(byteArray, np.uint8)  # npArray is
an array of integers

        # Do something cool
        npArray = cv2.Canny(npArray, 350, 550)

        # Reshape array into an RGB image matrix of shape
(channels, rows, cols)
        rows = val['rows']
        cols = val['cols']
        channels = val['channels']
        imgR = npArray[0::4].reshape((H, W))
        imgG = npArray[1::4].reshape((H, W))
        imgB = npArray[2::4].reshape((H, W))
        img = np.stack((imgR, imgG, imgB))
        img = np.moveaxis(img, 0, -1)
        success, a_numpy = cv2.imencode('.jpg', img)
        a = a_numpy.tostring()
        count += 1
        print('Feed 3: Displaying packet {}'.format(count))
        yield (b'--frame\r\n'
               b'Content-Type: image/jpg\r\n\r\n' + a +
b'\r\n\r\n')


def get_stream4():
    print('Listening to Feed 4...')
    count = 0
    for msg in consumer4:
        # Conversion: base-64 string --> array of bytes --> array
of integers
        val = msg.value
        base64string = val['pix']  # pix is base-64 encoded string
```

```
        byteArray = base64.b64decode(base64string)  # byteArray is
an array of bytes
        npArray = np.frombuffer(byteArray, np.uint8)  # npArray is
an array of integers

        # Reshape array into an RGB image matrix of shape
(channels, rows, cols)
        rows = val['rows']
        cols = val['cols']
        channels = val['channels']
        imgR = npArray[0::4].reshape((H, W))
        imgG = npArray[1::4].reshape((H, W))
        imgB = npArray[2::4].reshape((H, W))
        img = np.stack((imgR, imgG, imgB))
        img = np.moveaxis(img, 0, -1)

        # Do something cool with RGB image
        gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
        ret, thresh = cv2.threshold(gray, 0, 255,
cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)

        # noise removal
        kernel = np.ones((3, 3), np.uint8)
        opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel,
iterations=2)
        # sure background area
        sure_bg = cv2.dilate(opening, kernel, iterations=3)
        # Finding sure foreground area
        dist_transform = cv2.distanceTransform(opening,
cv2.DIST_L2, 5)
        ret, sure_fg = cv2.threshold(dist_transform, 0.7 *
dist_transform.max(), 255, 0)
        # Finding unknown region
        sure_fg = np.uint8(sure_fg)
        unknown = cv2.subtract(sure_bg, sure_fg)

        # Marker labelling
        ret, markers = cv2.connectedComponents(sure_fg)
        # Add one to all labels so that sure background is not 0,
but 1
        markers = markers + 1
        # Now, mark the region of unknown with zero
        markers[unknown == 255] = 0

        markers = cv2.watershed(img, markers)
        img[markers == -1] = [255, 0, 0]

        success, a_numpy = cv2.imencode('.jpg', img)
        a = a_numpy.tostring()
        count += 1
        print('Feed 4: Displaying packet {}'.format(count))
        yield (b'--frame\r\n'
```

```
                b'Content-Type: image/jpg\r\n\r\n' + a +
b'\r\n\r\n')


if __name__ == "__main__":
    app.run(host='0.0.0.0', port='5001', debug=False)
```