НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«На правах рукопису»
УДК 004.6

До захисту допущено:
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютеризованих систем»

зіспеціальності 121«Інженерія програмного забезпечення»

на тему: «Математичне та програмне забезпечення системи виявлення аномалій у роботі морських суден»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-92мп
Шуліков ДмитроДмитрович _____

Науковий керівник:
доц., к.т.н.,
Баклан Ігор Всеволодович _____

Рецензент:
доц., к.т.н.,
Ткач Михайло Мартинович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма - «Інженерія програмного забезпечення комп'ютеризованих систем»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Шулікову Дмитру Дмитровичу

1. Тема дисертації «Математичне та програмне забезпечення системи виявлення аномалій у роботі морських суден»,науковий керівник дисертаціїБаклан Ігор Всеволодович, проф., д.т.н.,затверджені наказом по університету від 26 жовтня 2020 р. №3132-с

2. Термін подання студентом дисертації _____

3. Об'єкт дослідженняМетрики правильної роботи морських суден, зокрема їх двигунів

4. Вхідні дані Технічне завдання

5. Перелік завдань, які потрібно розробити виконати огляд метрик, за якими можна характеризувати нормальну роботи морських суден; виконати огляд даних, які потрібні для побудови метрик; обрати ряд алгоритмів та методів побудови метрик; розробити програмну реалізацію алгоритмів; провести аналіз отриманих результатів.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу

1) Схема двигуна внутрішнього згоряння

7. Орієнтовний перелік публікацій 1 публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Графічний	доц. Ліщук К.І.		

9. Дата видачі завдання_____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вивчення рекомендованої літератури	06.09.2020	
2	Аналіз метрик нормальноїроботи суден	06.09.2020	
3	Аналіз підходів до архітектури	07.09.2020	
4	Постановка та формалізація задачі	12.09.2020	
5	Дослідження існуючих алгоритмів для побудови метрик	22.09.2020	
6	Моделювання програмного забезпечення	20.10.2020	
7	Розробка програмного забезпечення	01.11.2020	
8	Виконання графічних документів	25.11.2020	
9	Оформлення дисертації	25.11.2020	
10	Подання МД на попередній захист	27.11.2020	
11	Подання МД рецензенту		
12	Подання МД на основний захист	18.12.2020	

Студент Шуліков Д.Д.

Науковий керівник Баклан І.В.

РЕФЕРАТ

Актуальність теми:Морські судна дуже давно є надважливою ланкою у товарних перевезеннях. Великою часткою витрат для власників такого бізнесу є технічне обслуговування суден. Варто зазначити, шо такі витрати часто є не раціональними, тому що для виявлення відхилень у роботі судна потрібно заїхати на технічне обслуговування, навіть якщо ніяких відхилень знайдено не буде, а обслуговування буде коштувати великих сум.

Мета дослідження: Підвищення ефективності моніторингу нормальної роботи морських суден.

Завдання дослідження:

- огляд метрик нормальної роботи суден;
- огляд алгоритмів для побудови метрик;
- вивчення наявних інструментів для аналізу даних у реальному часі;
- розробка системи для аналізу і обробки даних у реальному часі;
- моделювання та конструювання сервісу побудови метрик;
- розробка системи імітації реальних даних для тестування.

Об’єкт дослідження:Метрики нормальної роботи морських суден, зокрема їх двигунів.

Предмет дослідження:Інформативність побудованих метрик для виявлення аномалій у роботі морських суден.

Наукова новизна:

Нові можливості для побудови метрик на віддалених пристроях за рахунок використання сучасних інструментів для обробки великих об’ємів потокових даних.

Практичне значення отриманих результатів значно зменшена вартість технічного обслуговування морських суден, за рахунок уникнення надлишкових зупинок і перевірок.

Зв’язок з науковими програмами, планами, темами: робота виконувалась на кафедрі автоматизованих систем обробки інформації і

управління Національного технічного університету України "Київський політехнічний інститут імені Ігоря Сікорського".

Публікації: наукові положення дисертації опубліковано в Шуліков Д.Д.Баклан І.В.Вибір платформи для обробки потокових даних з сенсорів морських суден / Шуліков Д.Д., Баклан І.В. // Матеріали V Всеукраїнської науково-практичної конференції молодих вчених та студентів «Інформаційні системи та технології управління»(ІСТУ-2020) - м. Київ: НТУУ “КПІ ім. Ігоря Сікорського”, 26-27 листопада 2020р.

КЛЮЧОВІ СЛОВА: МОРСЬКІ СУДНА, ВЕЛИКІ МАСИВИ ДАНИХ, ПОТОКОВІ ДАНІ

ABSTRACT

Topicality: Sea vessels is the important link in freight transport. A large part of the costs for the owners of such businesses is the maintenance of ships. It is worth noting that such costs are often unreasonable, because to detect deviations in the operation of the vessel you need to call for maintenance. This maintenance will cost large sums even if no deviations are found.

The aim of the study: Improving the efficiency of monitoring the normal operation of sea vessels.

Tasks of the study:

- analysis of metrics of normal operation of vessels;
- analysisof algorithms for constructing metrics;
- study of available tools for real-time data analysis;
- development of a system for analysis and data processing in real time;
- modeling and construction of metrics construction service;
- development of a system for simulating real data for testing.

Object of study: Metrics of normal operation of sea vessels, in particular their engines.

Subject of research: Informativeness of the constructed metrics for detection of anomalies in work of sea vessels.

Scientific novelty:

New capabilities for building metrics on remote devices through the use of modern tools for processing large amounts of streaming data.

The practical value of the obtained resultssignificantly reduced the cost of maintenance of ships, by avoiding unnecessary stops and inspections.

Relationship with scientific programs, plans and themes: The work was carried out at the Department of Automated Information Processing and Management Systems of the National Technical University of Ukraine “Igor Sikorsky Kyiv Polytechnic Institute”.

Publications: Scientific provisions of the dissertation published in Shulikov D.D. Baklan I.V. The choice of platform for processing streaming data from sensors of sea vessels / Shulikov D.D., Baklan I.V. // Proceedings of the Fifth All-Ukrainian Scientific and Practical Conference of Young Scientists and Students "Information Systems and Management Technologies" (ISTU-2020) - Kyiv: NTUU “KPI them. Igor Sikorsky”, November 26-27, 2020.

KEY WORDS: SEA VESSELS, LARGE DATA SETS, STREAMING DATA

ЗМІСТ

ПЕРЕДІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

RDD – Еластичнірозподіленінабориданих

ACID - Атомарність, послідовність, ізоляція та довговічність

ETL – Витяг, перетворення та завантаження

UDAF – Визначенакористувачемагрегуючафункція

SOA – Сервісноорієнтованаархітектура

AtmostOnce – Не більше одного разу

AtleastOnce – Хоча б один раз

ExactlyOnce – Чітко один раз

RPM – кількість обертів за хвилину

ВСТУП

Все більше та більше з’являється технологій для світу інтернету речей. Інтернет речей - це назва для сукупності пристроїв з підтримкою мережі, за винятком традиційних комп’ютерів, таких як ноутбуки та сервери. Типи мережевих з’єднань можуть включати WiFi, блютуз та NFC.[1]Інтернет речей включає такі пристрої, як "розумні" прилади, наприклад холодильники та термостати, системи домашньої безпеки, комп’ютерна периферія, така як веб-камери та принтери,годинники тощо. Ці пристрої використовують Інтернет-протокол, той самий, який ідентифікує комп’ютери у всесвітній мережі та дозволяє їм спілкуватися між собою. Метою Інтернету речей є наявність пристроїв, які самостійно збирають дані у реальному часі, покращуючи ефективність та виводячи важливу інформацію на поверхню швидше, ніж система, яка залежна від втручання людини.[1]

Інтернет речей відкриває потенціал для мільярдів підключених смарт-пристроїв для спілкування між собою. IoT дозволяє практично будь-якій системі використовувати Інтернет та екосистему хмарних обчислень, щоб впроваджувати інновації та робити різноманітний діапазон об'єктів розумнішими та обізнанішими. Зростання кількості підключених до Інтернету пристроїв обумовлено широкою доступністю високоякісних, надійних бездротових зв’язків, а також низькою вартістю вбудованих компонентів.

Хоча пристрої IoT мають багато різних форм-факторів, всі вони потребують загальних функціональних будівельних блоків, таких як обробка та захист, зондування та спрацьовування, підключення, кондиціонування сигналу, захист, управління енергією та енергією.

Для магістерської дисертації була обрана тема з домену морських суден, тому що організація архітектура такої системи має суттеві складнощі, зокрема через відсутність постійного з’єднання з мережею, то ж було цікаво вирішити таку проблему.

Сьогодні моніторинг ефективності морських суден є ключовим інструментом не тільки для капітана та екіпажу, а й для судновласників та менеджерів флоту. Обсяг даних, який може бути зібраний та його використання, важко переоцінити. Судновласники та судноплавні компанії можуть використовувати таку інформацію для прогнозування витрат і часу на технічне обслуговування, оптимізації споживання палива або побудови найкоротших маршрутів між портами. У той же час капітан може використовувати таку інформацію, щоб оптимізувати спосіб керування судном. Але збір такої інформації та, що найважливіше, її аналіз - це справді складна частина моніторингу ефективності.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Постановка задач дослідження

Метою дослідження є підвищення ефективності моніторингу нормальної роботи морських суден.

Призначення роботи у данній дисертації є створення системи потокової обробки даних з датчиків морських суден для побудови метрик нормальної роботи суден.

Для досягнення поставленної мети необхідно вирішити наступні завдання:

- огляд метрик нормальної роботи суден;
- огляд алгоритмів для побудови метрик;
- вивчення наявних інструментів для аналізу даних у реальному часі;
- розробка системи для аналізу і обробки даних у реальному часі;
- моделювання та конструювання сервісу побудови метрик;
- розробка системи імітації реальних даних для тестування.

1.2 Обробка великих об’ємів потокових даних

Згідно з нещодавнім звітом IBM, «Дев'яносто відсотків даних у світі було створено лише за останні два роки. З появою нових пристроїв, датчиків та технологій, зростання даних прискориться ще більше».

Технічно це означає, що обробка великих масивів даних стане складнішою. Багато випадків (наприклад, оголошення для мобільних додатків, виявлення шахрайства, бронювання, моніторинг пацієнтів тощо) потребують обробки даних у режимі реального часу для швидкого прийняття рішень. Ось чому розподілена потокова обробка стала дуже популярною у світі великих даних.

Сьогодні доступна низка фреймворків для потокової обробки з відкритим кодом. Цікаво, що майже всі вони досить нові і були розроблені лише за

останні кілька років, тож новачку досить легко заплутатися в розумінні цих платформ.

Найелегантніше визначення, яке я знайшов, це "тип механізму обробки даних, який розроблений з урахуванням нескінченних наборів даних".

На відміну від пакетної обробки, де дані обмежуються початком і кінцем у роботі, а робота закінчується після обробки цих даних - потокова обробка призначена для опрацювання необмежених даних, що надходять у режимі реального часу, безперервно протягом днів, місяців та років.

1.2.1 Характеристики інструментів для обробкипотоковихданих

Існує кілька важливих характеристик та термінів, пов’язаних із обробкою потоку, про які ми повинні знати, щоб зрозуміти сильні сторони та обмеження будь-якої структури потоку:

- гарантія доставки (DeliveryGuarantees);
- відмовостійкість (FaultTolerance);
- управління станом (StateManagement);
- продуктивність (Performance);
- розширені функції;
- зрілість.

Гарантія доставки (DeliveryGuarantees) означає гарантію того, що незважаючи ні на що, певний вхідний запис у потоковому механізмі буде оброблений. Це може бути як atleast-once (буде оброблено принаймні один раз, навіть у разі відмов), atmost-once (може бути не оброблено у разі відмов) або exactly-once (буде оброблено один і рівно один раз, навіть у випадку відмов). Очевидно, що режим exactly-once найбажаніший, але його важко досягти в розподілених системах, потрібно знаходити компроміси з продуктивністю.

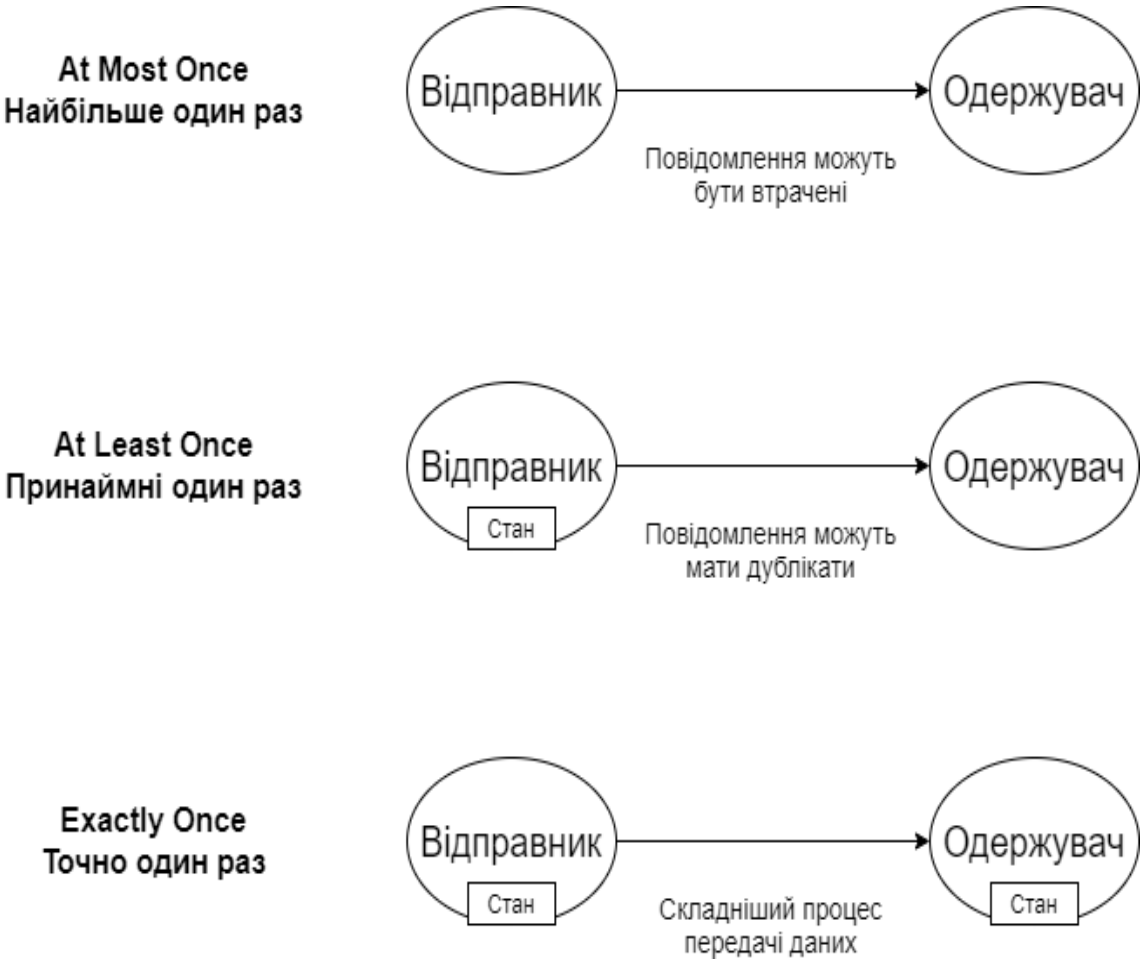


Рисунок 1.1 – Види гарантій доставки

Відмовостійкість (FaultTolerance) важливау разі збоїв, таких як відмови вузлів, відмови мережі тощо. Фреймворк повинен мати можливість відновлення та починати обробку знову з того місця, де він зупинився. Це досягається за допомогою регулярного збереження стану потокової обробки на постійне сховище.

Управління станом (StateManagement) грає рольу випадку вимог до обробки даних, де нам потрібно підтримувати деякий стан застосунку (наприклад, кількість входжень кожного окремого слова із записів).Фреймворк повинен мати можливість забезпечити якийсь механізм збереження та оновлення інформації про стан.

Продуктивність (Performance) включає затримку (як швидко запис може бути оброблений), пропускну здатність (оброблені записи / секунду) та масштабованість. Затримка повинна бути якомога меншою, тоді як пропускна

здатність повинна бути якомога більшою. Важко отримати обидва показники одночасно.

Розширені функції цефункції, які необхідні, якщо вимоги до обробки потоку є складними. Наприклад, обробка записів на основі часу, який був сформований у джерелі та агрегація даних у часових вікнах, що є дуже важливим функціоналом, тому що багато метрик у моїй роботі базуються на використанні даних за певний проміжок часу. Розрізняють декілька видів агрегацій на основі часових вікон:

- tumblingwindows - мають фіксований часовий розмір і не перекриваються;

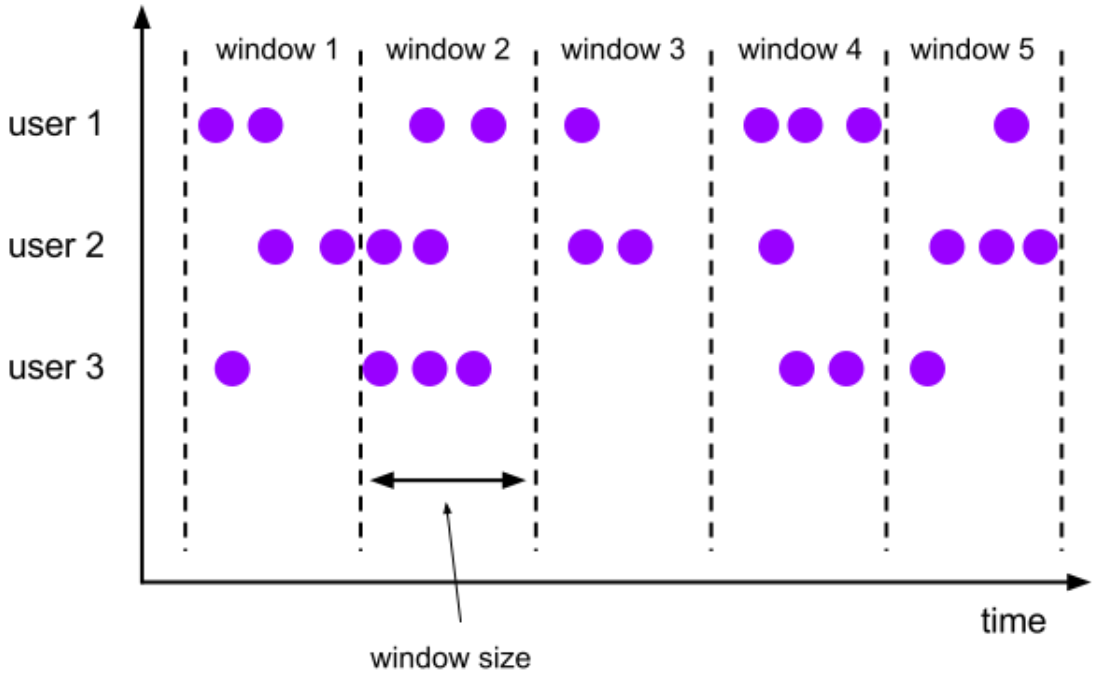


Рисунок 1.2 – Часові вікна з фіксованим розміром, які не перетинаються

- slidingwindows– часові вікна, з фіксованим розміром, що можуть перетинатись;

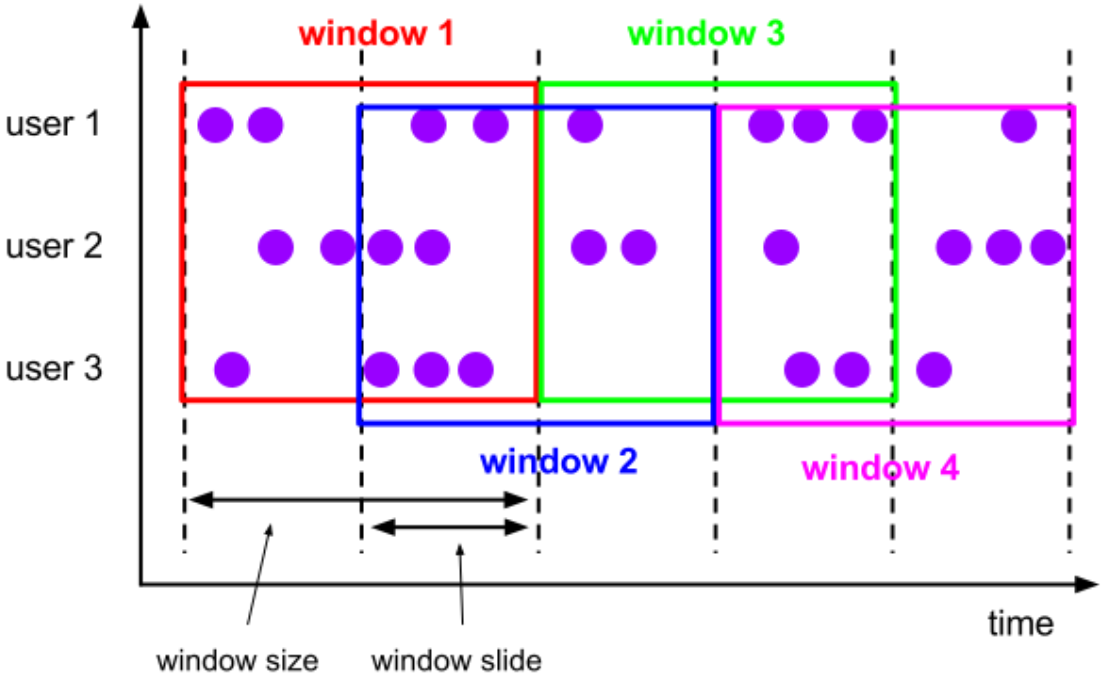


Рисунок 1.3 – Часові вікна з фіксованим розміром, що перетинаються

- sessionwindows – вікнане перекриваються і не мають фіксованого часу початку та закінчення. Натомість вікно сеансу закривається, коли воно не отримує елементів протягом певного періоду часу.

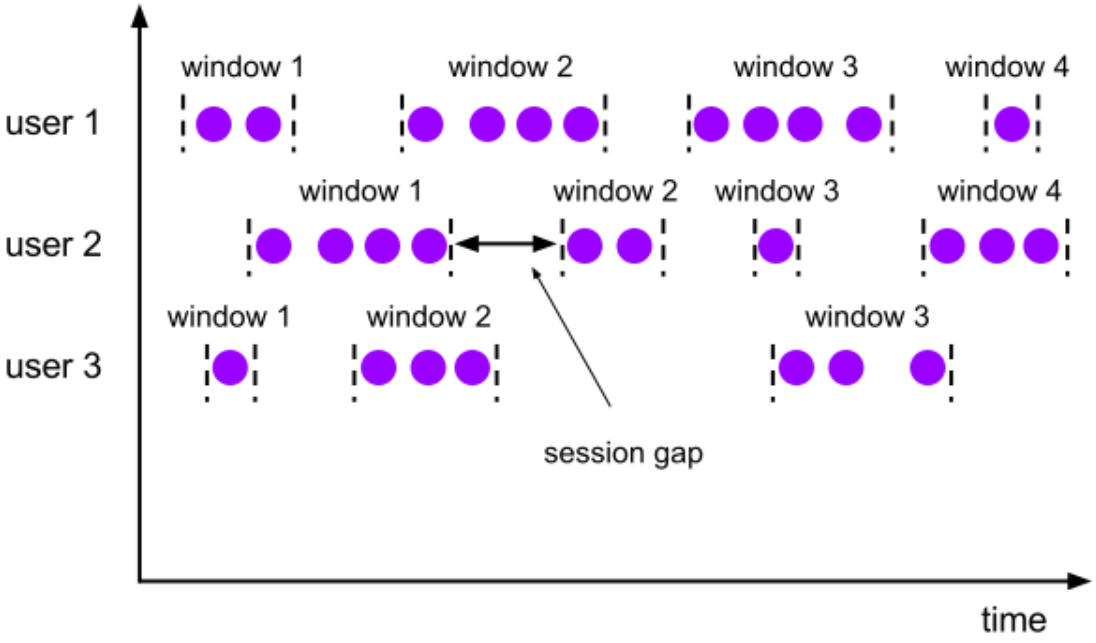


Рисунок 1.4 – Сесійні вікна

Зрілість підкреслює важливість того, що інструмент вже перевірений та випробуваний великими компаніями. Більше шансів отримати хорошу підтримку у мережі та допомогу на stackoverflow.

1.2.2 Типи обробки потокових даних

Знаючи терміни, які щойно були описані, тепер можна зрозуміти, що існує два підходи до впровадження обробки потоків даних:

Нативнийстрімінг - кожен вхідний запис обробляється як тільки він надходить, не чекаючи інших. Є кілька безперервно запущених процесів, які ми називаємо операторами, і кожен запис проходить через ці процеси для обробки. Приклади: Storm, Flink, KafkaStreams, Samza.

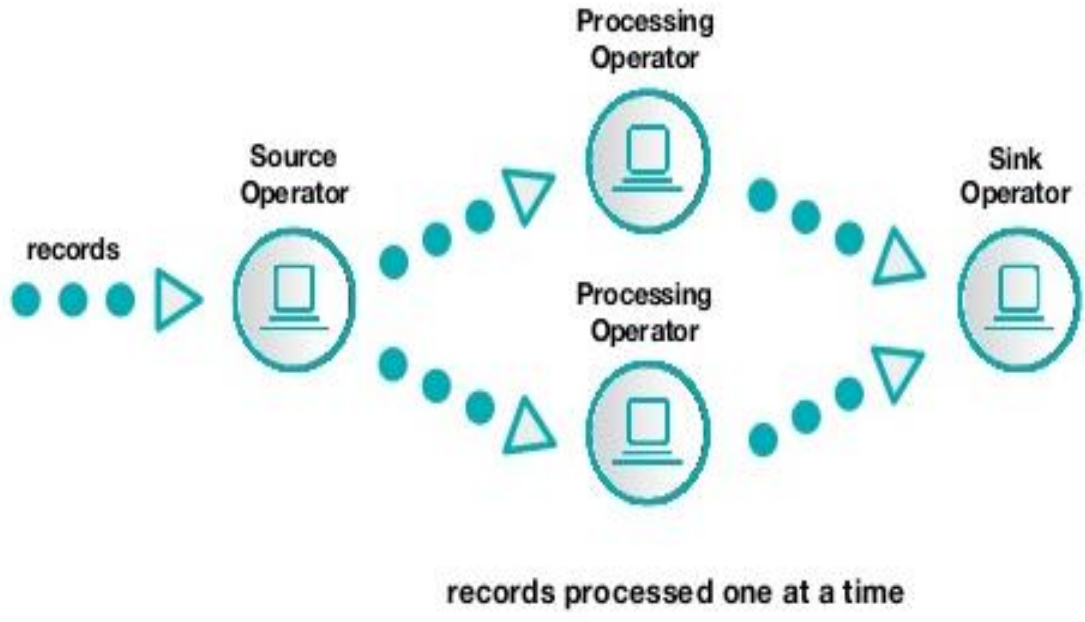


Рисунок 1.5 – Нативнийстрімінг

Мікробатчінг - означає, що вхідні записи через кожні кілька секунд групуються, а потім обробляються в одній міні-партії із затримкою в кілька секунд. Приклади: SparkStreaming, Storm-Trident.

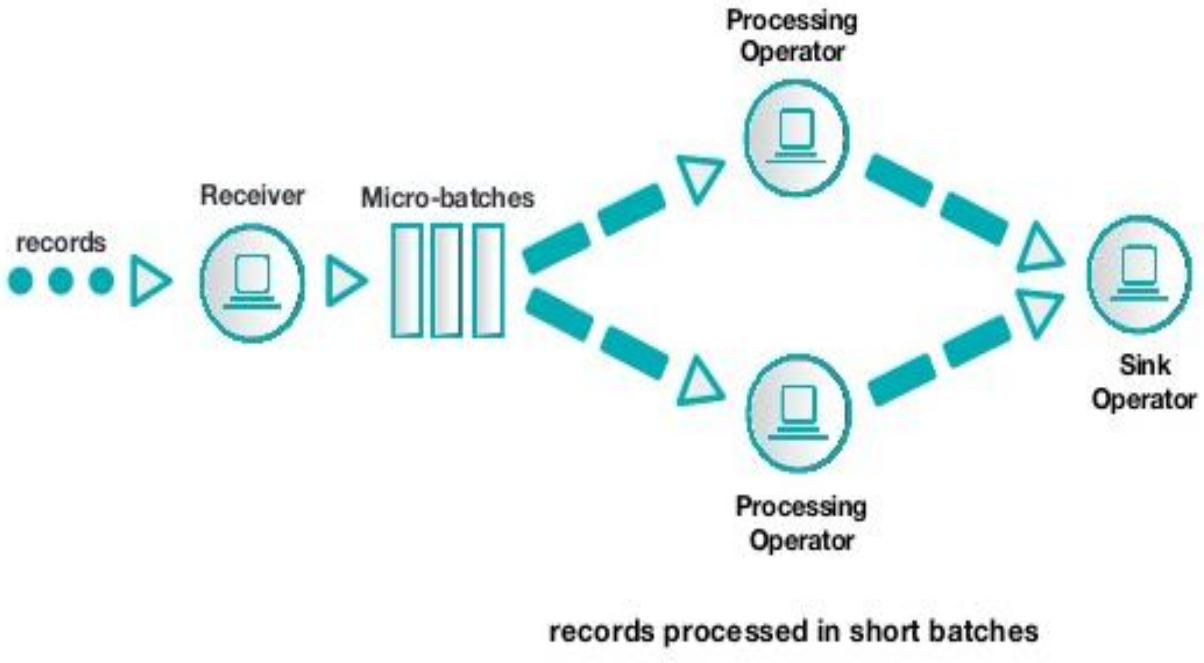


Рисунок 1.6 – Мікробатчінг

Обидва підходи мають деякі переваги та недоліки:

Нативнийстрімінг є більш природним, оскільки кожен запис обробляється відразу після надходження, що дозволяє фреймворку досягти мінімально можливої затримки. Але це також означає, що важко досягти відмовостійкості без шкоди для пропускної здатності, оскільки для кожного запису нам потрібно зберігати контрольну точку після обробки. Крім того, управління станом є простим, оскільки існують тривалі процеси, які можуть легко підтримувати необхідний стан.

Мікробатчінг навпаки, є цілком протилежним. Толерантність до несправностей надається безкоштовно, оскільки це, по суті, пакет. Пропускна здатність також велика, оскільки обробка та перевірка буде виконуватися одним пострілом для групи записів. Але це буде відбуватись з деякою затримкою і це не буде природний потік даних.

1.2.3 Фреймворкидляпотоковоїобробкиданих

Apache Storm - найстарішийфреймворкдляпотоковоїобробкизвідкритимкодоміодинізнайбільшзрілихтанадійних. Ценативнапотоковаобробка. Підходить для простихвипадківприйняттярішень на основіподій.

Переваги:

- дуже низька затримка;
- висока пропускна здатність;
- зрілість;
- нативна потоковаобробка;
- відмінно підходить для нескладних випадків використання потокових даних.

Недоліки:

- відсутність управління станом;
- відсутність розширених функцій, таких як обробка часу подій, агрегування, часові вікна, сеанси тощо;
- тільки режим atleast-once для обробки даних.

SparkStreaming - Spark виявився справжнім спадкоємцем hadoop в пакетній обробці та першим фреймворком, який повністю підтримує лямбда архітектуру (де використана як пакетна, так і потокова обробка; пакетна для коректності, потокова для швидкості). Цей фреймворк дуже популярний, зрілий і широко прийнятий. SparkStreaming безкоштовно надається разом із Spark і використовує мікробатчінг для потокової обробки.

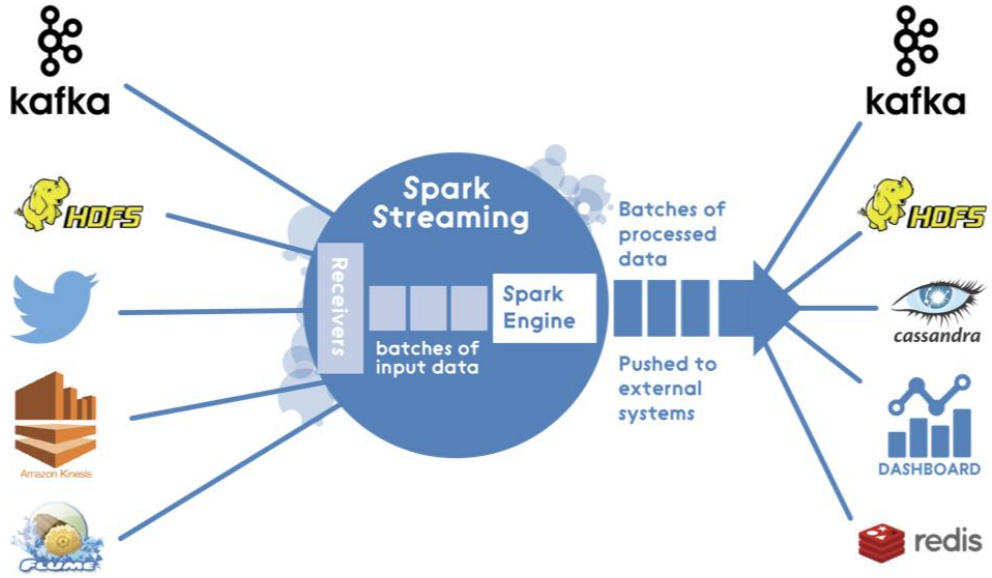


Рисунок 1.7 – Мікробатчінгова архітектура sparkstreaming

До випуску 2.0 SparkStreaming мав деякі серйозні обмеження продуктивності, але з новим випуском 2.0+ він називається структурованою потоковою обробкою і оснащений багатьма додатковими функціями, такими як користувацьке управління пам'яттю (аналог flink), водяними знаками, підтримкою обробки часу подій тощо. Також структурований потік є набагато абстрактнішим, і є можливість переключення між мікро-пакетним режимом та режимом безперервної потокової обробки у випуску 2.3.0. Режим безперервного потокової обробки обіцяє зменшити затримку у сторону Storm та Flink, але він все ще перебуває в початковій стадії розробки з багатьма обмеженнями в операціях.

Переваги:

- відсутність управління станом;
- підтримка лямбда архітектури;
- висока пропускна здатність;
- відмовостійкість за замовчуванням через мікро-пакетну природу;
- простий у використанні API вищого рівня;
- велика спільнота розробників;
- exactlyoncережим.

Недоліки:

- несправжнійстрімінг, не підходить для вимог із низькою затримкою;
- дуже багато параметрів для налаштування;
- відсутність стану;
- відстає від Flink у багатьох розширених функціях.

Fink, як і Спарк, також має академічний бекграунд. Спарк прийшов з UC Berkley, Флінк - з Берлінського університету TU. Як і Spark, він також підтримує лямбда-архітектуру. Але реалізація повністю протилежна реалізації Spark. Flink - це, по суті, справжній потоковий процесінг. Хоча API в обох фреймворках схожі, але вони не мають жодної подібності в реалізаціях. У Flink кожна функція, така як map, filter, reduce тощо реалізована як тривалий оператор (аналогічно Storm).

Flinkмає кластерну архітектуру, де присутні:

- клієнти, щонадаютьзадачі для виконання;
- джоб менеджер, що розподіляє роботи поміж таскменеджерів;
- таскменеджери, щобезпосередньовиконують роботу.

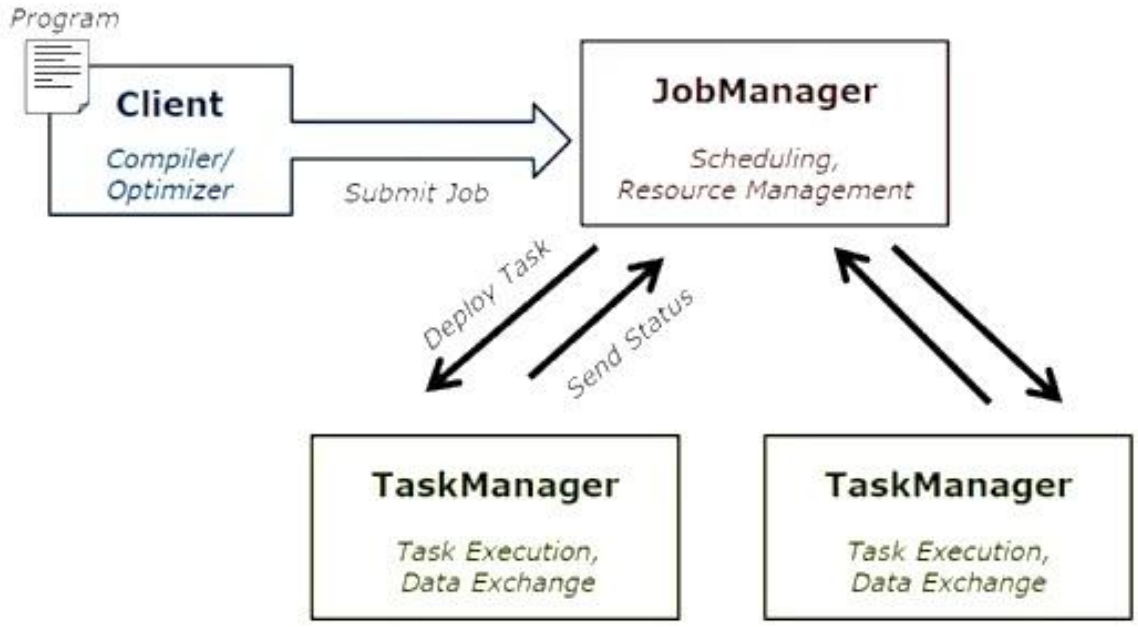


Рисунок 1.8 – Кластерна архітектура Flink

Також Flinkмає дуже зручний механізм для відновлення роботи після аварійного завершення, що зветься чекпойнтами, коли кожен певний період часу створюються знімки стану системи на даний момент.

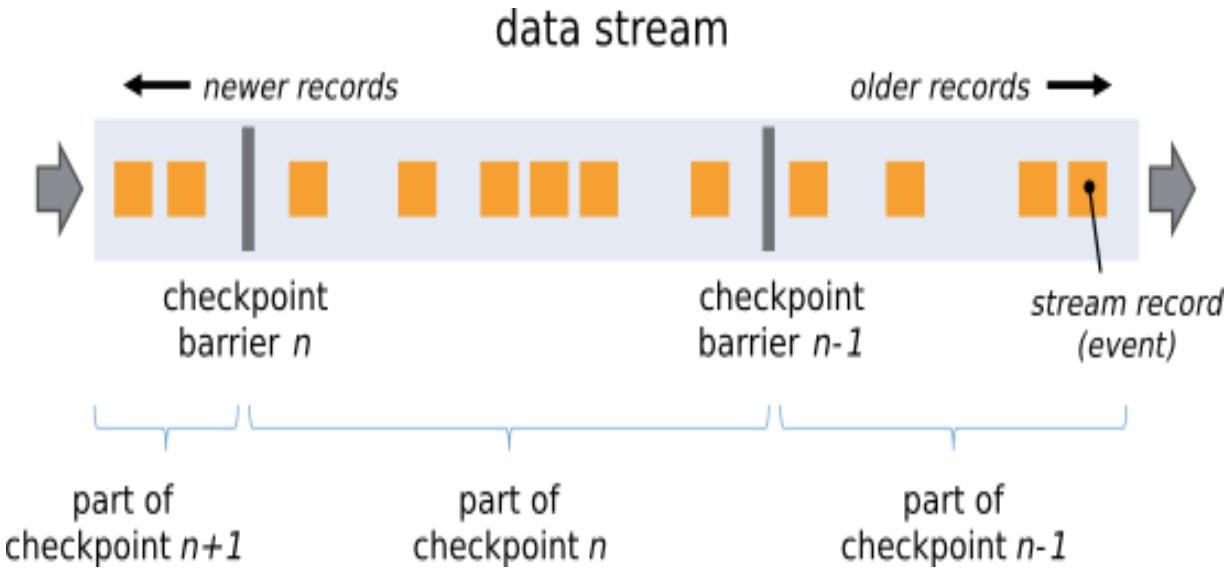


Рисунок 1.9 – Механізм чекпойнтів у Flink

Переваги:

- лідер інновацій у потоковій обробці даних з відкритим кодом;
- перший фреймворк нативноїпотокової обробки з усіма розширеними функціями, такими як обробка часу подій, водяні знаки тощо;
- низька затримка з високою пропускною здатністю, що налаштовується відповідно до вимог;
- авторегулювання, не надто багато параметрів для налаштування;
- широке визнання таких великих компаній, як Убер, Алибаба;
- exactlyoncережим.

Недоліки:

- спільнота не така велика, як Спарк, алешвидко зростає;

- тільки потокова обробка, на відміну від Спарк.

KafkaStreams, на відміну від інших фреймворків, невелика бібліотека для потокової обробки даних KafkatoKafka. Не має відповідності з точки зору продуктивності з Flink, але також не потребує окремого кластера для запуску. Дуже зручний і простий фреймворк у розгортанні та початку роботи. Однією з головних переваг KafkaStreams є те, що обробка повідомлень відбувається чітко один раз (exactlyonce). Це можливо, оскільки джерелом, а також місцем призначення є Kafka, а з версії Kafka 0.11, випущеної приблизно в червні 2017 року, підтримується exactlyonce семантика. Щоб увімкнути цю функцію, нам просто потрібно увімкнути флаг у налаштуваннях.

Переваги:

- легковісна бібліотека, що є добре для мікросервісів, додатків IOT;
- не потребує виділеного кластера;
- успадковує всі корисні характеристики кафки;
- підтримує з'єднання потоків;
- внутрішньо використовує rocksDb для підтримки стану;
- exactlyoncережим.

Недоліки:

- інтеграція тільки з кафкою;
- не для великих навантажень, таких як SparkStreaming, Flink.

Samza схожа на KafkaStreams. Є багато подібностей. Обидва ці фреймворки були розроблені тими самими розробниками, які впровадили Samza в LinkedIn, а потім заснували Confluent, де вони написали KafkaStreams. Обидві ці технології тісно пов'язані з Kafka: беруть необроблені дані з Kafka, а потім повертають оброблені дані назад до Kafka. Самза - це свого роду масштабована версія KafkaStreams. Поки KafkaStreams - це бібліотека, призначена для мікросервісів, Samza - це повноцінна кластерна обробка.

Переваги:

- добре підтримує великі стани інформації (добре для використання у випадку приєднання потоків) за допомогою журналів rocksDb та kafka;
- толерантність до несправностей та високапродуктивністьізвикористаннямвластивостейKafka;
- низька затримка та висока пропускна здатність
- зрілість.

Недоліки:

- інтеграція тільки з кафкою;
- відсутність розширених функцій.

1.2.4 Вибір найкращого фреймворку

Важливо мати на увазі, що жоден механізм обробки не може бути срібною кулею для кожного випадку використання. Кожен фреймворк має деякі сильні сторони та деякі обмеження. Якщо варіант використання простий, немає необхідності шукати найновіші та найкращі фреймворки, якщо це складно вивчити та впровадити. Багато залежить від того, скільки ми готові вкласти, скільки хочемо взамін. Наприклад, якщо це проста система сповіщення на основі подій IOT, з Storm або KafkaStreams цілком чудово працювати.

У той же час нам також слід усвідомлено розглянути, якими будуть можливі випадки використання в майбутньому? Чи можливо, що в майбутньому можуть виникнути вимоги до розширених функцій, таких як обробка часу подій, агрегування, приєднання потоків тощо? Якщо відповідь позитивна, тоді краще піти на вдосконалені фреймворки потокової обробки, такі як SparkStreaming або Flink. Після інвестування в одну технологію - її важко змінити пізніше. Якщо ми добре розуміємо сильні сторони та обмеження фреймворків разом із необхідними варіантами використання, тоді легше обрати або хоча б фільтрувати наявні варіанти. Нарешті, завжди добре мати POC, коли буде обрано пару варіантів.

Простір ApacheStreaming розвивається настільки швидкими темпами, що ця дисертація може застаріти з точки зору інформації вже через пару років. В даний час Spark і Flink є важкоатлетами, що знаходяться попереду, з точки зору розвитку подій, але новий учасник все ще може прийти і приєднатися до гонки.

1.3 Архітектурні вимоги до розробки програмного забезпечення

1.3.1 Загальна архітектура

Розуміючи системи потокових даних, тепер можна звернути свою увагу на архітектурний план, який буде використовуватись для реалізації завдання дисертації:

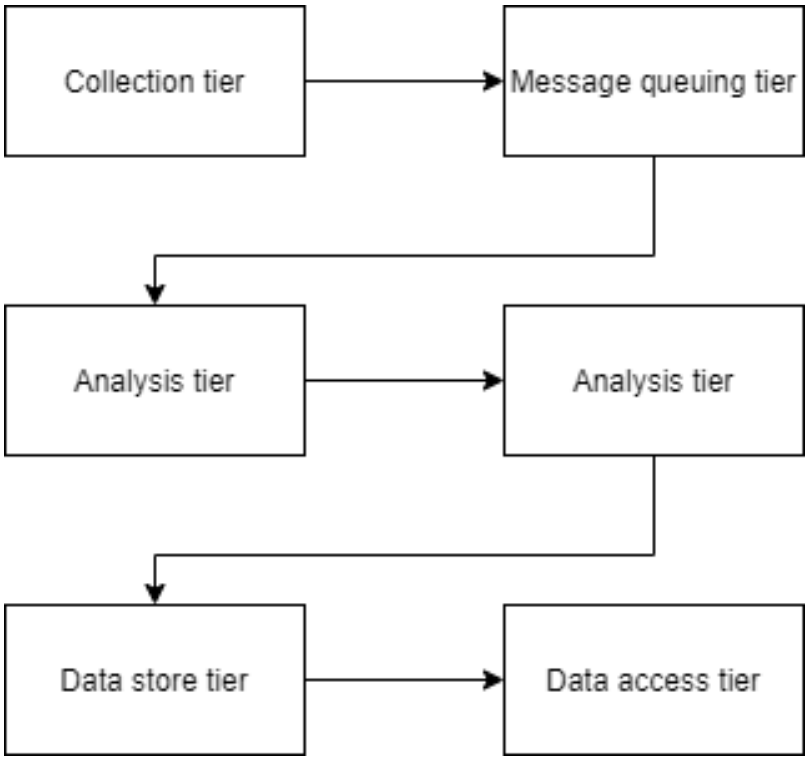


Рисунок 1.10 – Узагальнена архітектура потоку даних

На рисунку 1.10можнавиділитинаступнірівні архітектури:

- рівень збору інформації, у моєму випадку з сенсорів;
- рівень черг повідомлень, для підтримання гнучкої обробки;

- рівень аналізу даних, де буде використовуватись один з інструментів для обробки потокових даних, розглянутих у попередньому розділі;
- рівень зберігання даних;
- рівень доступу до даних, для відображення порахованих метрик.

1.3.2 Рівеньчергиповідомлень

На перший поглядможездатись, щоцейрівень не обов’язковий у запропонованій архітектурі, яку можна спростити прибравши його та передаючи дані напряму до рівня аналізу.

Звичайно, можна створити цілу систему потокового передавання на одній машині, і кожен рівень буде безпосередньо викликати наступний, але,зазвичай, потокова система буде охоплювати багато машин. При розробці програмної системибажаною якістю, до якої слід прагнути, є роз'єднання різних компонентів. Такий підхід відомий як мікросервісна архітектура. З потоковою системою ми хочемо того ж: роз’єднати компоненти на рівні.

В основі потокової системи, як і будь-якої розподіленої системи, лежить зв'язок між численними машинами. Якщо подивитись на міжпроцесорне спілкування в літературі, можна знайти численні моделі; у дисертації основна увага приділяється моделі масового обслуговування повідомлень. Застосувавши цю модель, рівень збору даних буде відокремлений від рівня аналітики. Це роз'єднання дозволяє рівням працювати на вищому рівні абстракції, передаючи повідомлення та не маючи явних викликів до наступного рівня. Це дві хороші властивості, які має будь-яка система, не кажучи вже про розподілену потокову. Таке роз’єднання надає чудові переваги.

Основними компонентами рівня черги повідомлень євиробник, брокер та споживач.

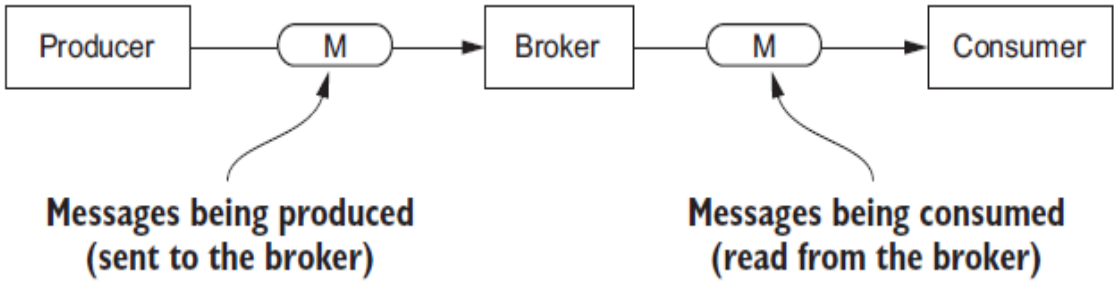


Рисунок 1.11 – Основні компоненти черги повідомлень

Виробник і споживач виконують роботу, яка повністю відповідає їх іменам: виробник виробляє повідомлення, а споживач споживає повідомлення. Потрібно звернути увагу на рисунок1.11, девикористовується термін«брокер», а не «черга повідомлень». Брокер це дуже важлива абстракція, оскільки брокер може керувати кількома чергами. Малюнок 1.12 показує, що черга повідомлень існує, але абстрагована брокером.

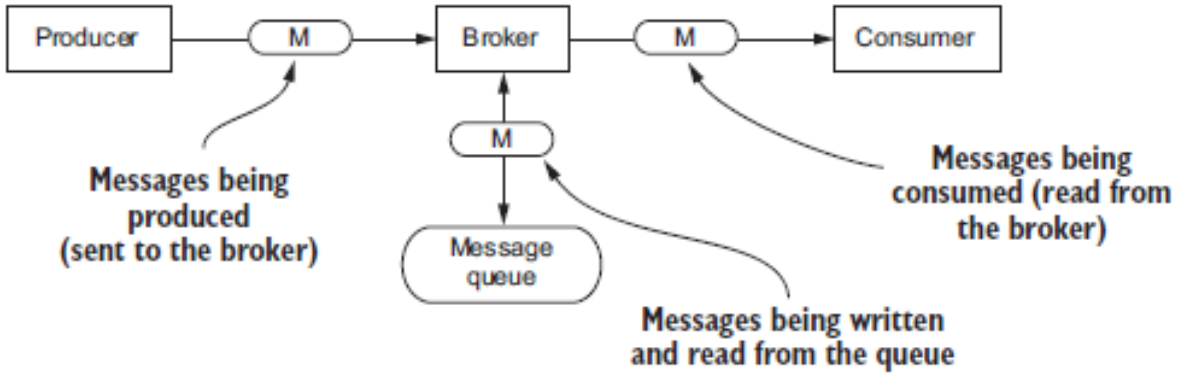


Рисунок 1.12 – Брокер з чергою повідомлень

Тепер до рисунку 1.10 із загальною архітектурою можна додати щойно описані деталі:

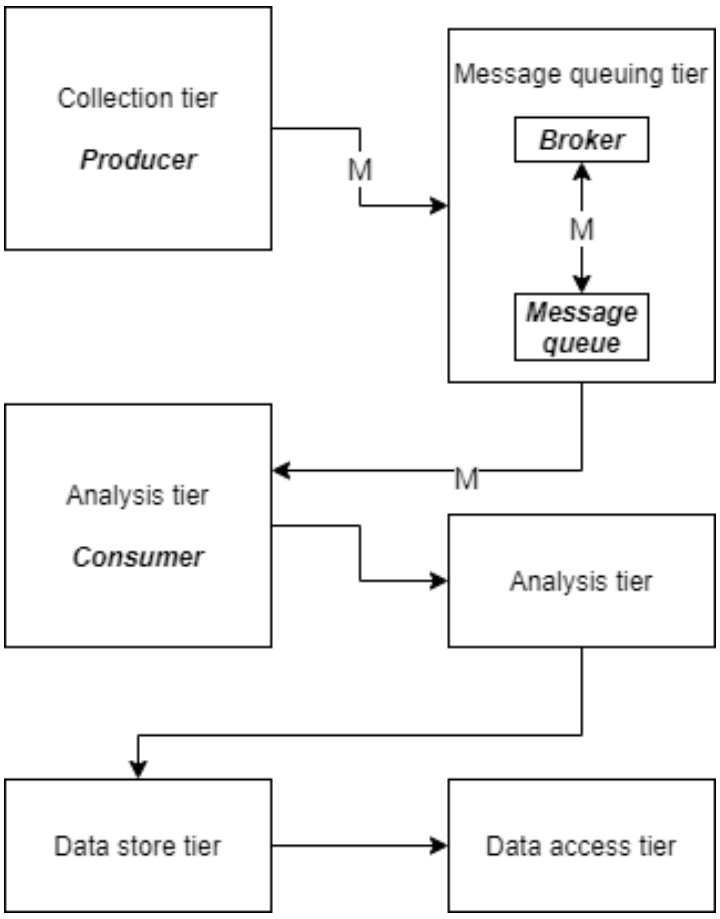


Рисунок 1.13 – Загальна архітектура з деталями рівня черги повідомлень

Ціль рівня черги повідомлень - розділити різні рівні системи. Рівень черги повідомлень дозволяє робити це з рівнями збору даних та аналізу. Це потрібно через декілька причин.

Залежно від потреб бізнесу та дизайну потокової системи, можна опинитися в ситуації, коли виробник (рівень збору) генерує повідомлення швидше, ніж споживач (рівень аналізу) може їх споживати. Часто це пов’язано з тим, що рівень аналізу вимагає більшого часу обробки, ніж рівень збору, і, можливо, він не зможе обробити дані так швидко, що створює таку проблему, що має термін backpressure.

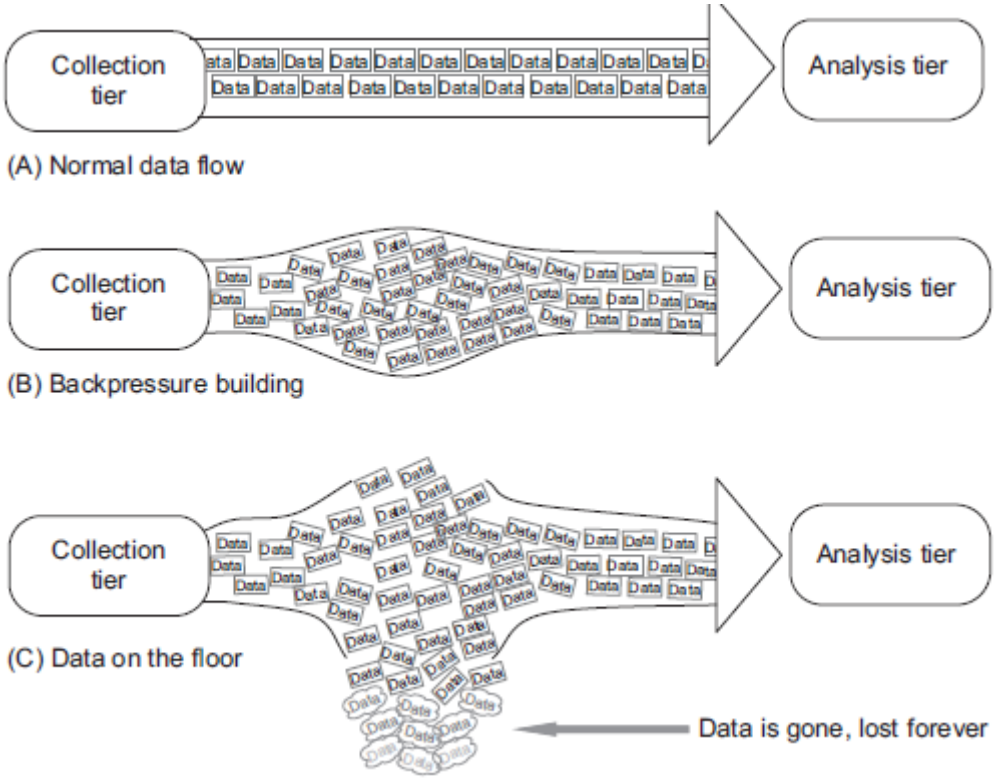


Рисунок 1.14 – Ілюстрація backpressure

Розглянемо три етапи на рисунку 1.14:

- крок А - виглядає цілком нормально, і це те, що хотілося б бачити;
- крогБ - можнасказати, щощось не зовсім правильно - зворотнийтискнаростає;
- крок С- канал передачі даних зруйновано під тиском і дані зникають назавжди.

Це погана ситуація, оскільки зараз втрачаються дані, а для деяких підприємств це може бути катастрофічним. Спочатку можна подумати, що це проблема споживача, і все, що потрібно зробити, це додати більше споживачів або зробити їх швидшими. Але це зовсім не проблема споживача; у багатьох випадках використання, цілком прийнятно для споживачів повільно читатидані або час від часу бути поза мережею, офлайн. Наприклад, споживачі можуть захотіти прочитати всі повідомлення лише щогодини, щоб підтримати варіант використання пакетної обробки; вони час від часу будуть в автономному режимі, а потім підключатимуться та читатимуть дані за останню годину.

Потрібно пам’ятати, що не всі системи масового обслуговування повідомлень забезпечують цей тип контролю потоку виробників, залишаючи зарозробником додатків, контроль за швидкістю, з якоюрівень колекції видає повідомлення. Можливість підтримувати повільнне читання повідомлень зі сторони споживача або періодичне перебуванням у режимі офлайн, забезпечується продуктами, що чергують повідомлення та підтримують довготривалі повідомлення.

Два популярних інструменти для організації рівня черги повідомлень, що підтримують асинхнонний обмін повідомленнями – ApacheKafkata RabbitMQ.

Асинхронний обмін повідомленнями - це схема обміну повідомленнями, при якій виробництво повідомлень виробником відділяється від обробки споживачем. Маючи справу з системами обміну повідомленнями, зазвичай виділяють дві основні схеми обміну –створення черг повідомлень та публікація / підписка.

У схемі спілкування з чергуванням повідомлень черги тимчасово відокремлюють виробників від споживачів. Кілька виробників можуть надсилати повідомлення в одну чергу; однак, коли споживач обробляє повідомлення, воно блокується або вилучається з черги і стає недоступним. Лише один споживач споживає певне повідомлення.

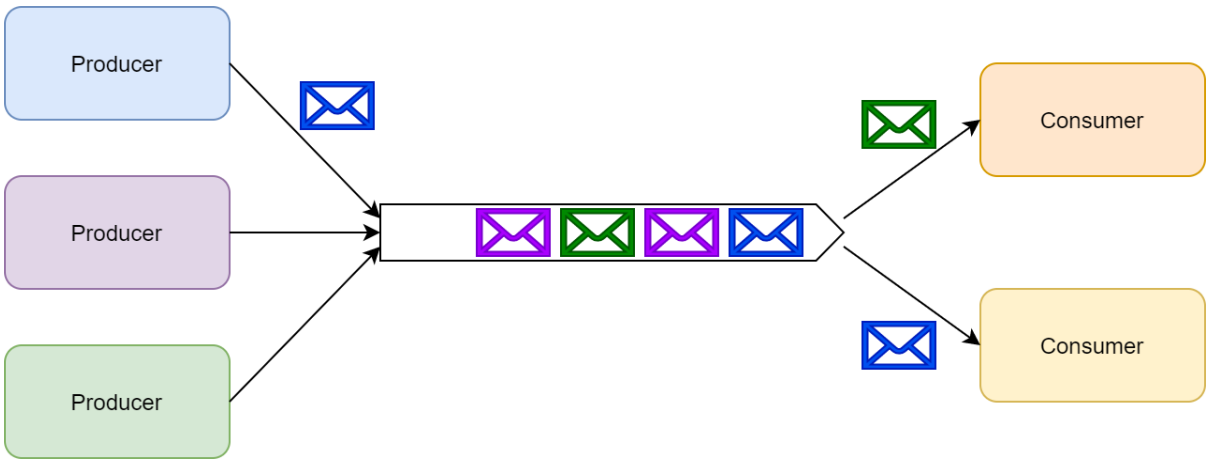


Рисунок 1.15 – Черга повідомлень

Як додаткове зауваження, якщо споживач не може обробити певне повідомлення, платформа обміну повідомленнями зазвичай повертає

повідомлення в чергу, де воно доступне для інших споживачів. Окрім тимчасового роз'єднання, черги дозволяютьсамостійно масштабувати виробників та споживачів, а також забезпечуючи ступінь відмовостійкості проти помилок обробки.

У шаблоні спілкування публікація / підпискаодне повідомлення може одночасно отримувати та обробляти декілька підписників.

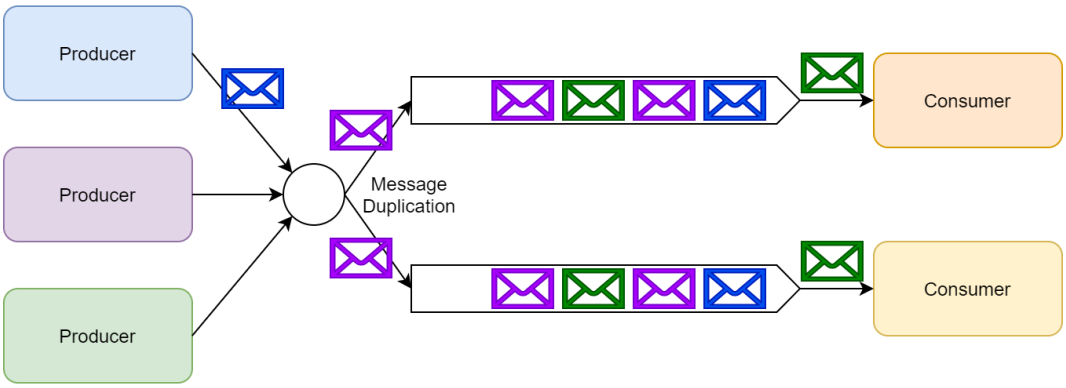


Рисунок 1.16 – Публікація / Підписка

Цей шаблон дозволяє видавцю, наприклад, повідомляти всіх підписників про те, що щось трапилося в системі.

Загалом існує два типи підписки: ефемерна та довготривала.

Ефемерна підписка активна лише до тих пір, поки споживач працює. Як тільки споживач вимикається, його підписка та повідомлення, які ще не обробились, втрачаються.

Довготривала підписка зберігається до тих пір, поки вона не буде явно видалена. Коли споживач вимикається, платформа обміну повідомленнями підтримує підписку, і обробка повідомлень може бути відновлена пізніше.

RabbitMQ є реалізацією посередника повідомлень, який часто називають шиною обслуговування. Він підтримує обидва описані вище шаблони обміну повідомленнями.

RabbitMQ підтримує класичні черги повідомлень «з коробки». Розробник визначає іменовані черги, і тоді видавці можуть надсилати повідомлення до цієї іменованої черги. Споживачі, в свою чергу, використовують одну і ту ж чергу для отримання повідомлень для їх обробки.

RabbitMQ реалізує pub / sub за допомогою біржи повідомленнь. Видавець публікує свої повідомлення на біржі повідомлень, не знаючи, хто є підписниками цих повідомлень.

Кожен споживач, який бажає підписатися на біржу, створює чергу, потім внутрішня реалізація біржі створює повідомлення для створених черг. Також можна фільтрувати повідомлення для деяких абонентів на основі різних правил маршрутизації.

Важливо зазначити, що RabbitMQ підтримує як короткочасні, так і довговічні підписки. Споживач може вирішити тип підписки, яку він хоче використовувати, за допомогою API RabbitMQ.

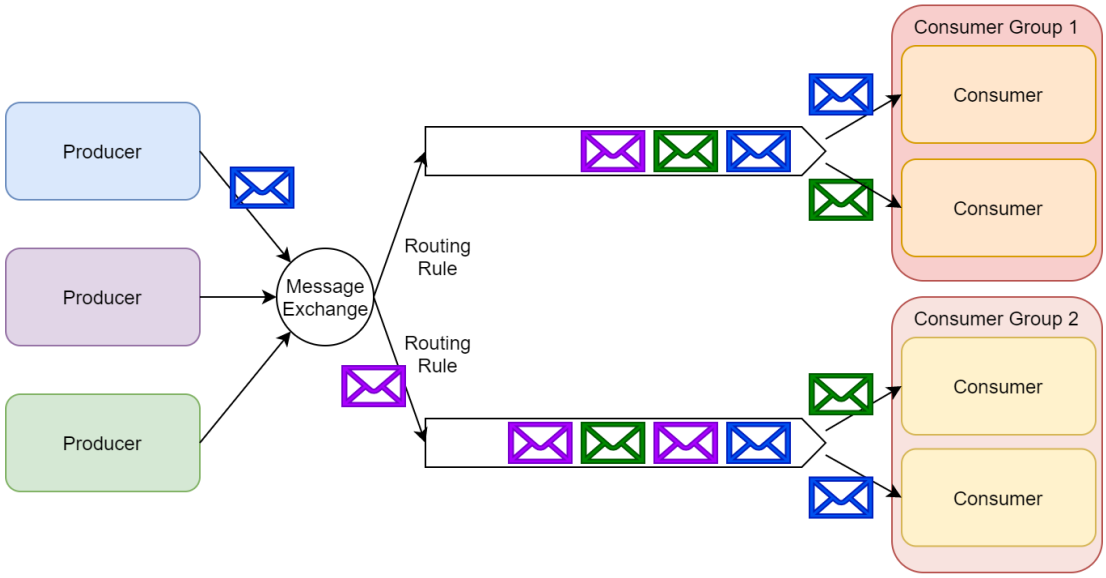


Рисунок 1.17 – Публікація / Підписка та побудова черг у комбінації

Завдяки архітектурі RabbitMQ ми також можемо створити гібридний підхід, що зображений на рисунку 1.17 - де деякі абоненти формують групи споживачів, які спільно обробляють повідомлення у формі конкуруючих споживачів за певною чергою. Таким чином, ми реалізуємо шаблон pub / sub,

одночасно дозволяючи деяким абонентам масштабувати обробку отриманих повідомлень.

ApacheKafka не є реалізацією брокера повідомлень. Натомість це розподілена потокова платформа.

На відміну від RabbitMQ, який базується на чергах та біржах, рівень зберігання Kafka реалізований за допомогою розділеного журналу транзакцій. Kafka також надає API для обробки потоків у реальному часі та API під'єднувачівдля легкої інтеграції з різними джерелами даних

Кафка не реалізує поняття черги. Натомість Кафка зберігає колекції записів у категоріях, що називаються темами.

Для кожної теми Кафка веде розділений журнал повідомлень. Кожен розділ - це впорядкована, незмінна послідовність записів, куди постійно додаються повідомлення.

Кафка додає повідомлення до цих розділів у міру їх надходження. За замовчуванням для рівномірного розповсюдження повідомлень по розділах він використовує круглий розділ.

Виробники можуть модифікувати цю поведінку для створення логічних потоків повідомлень.

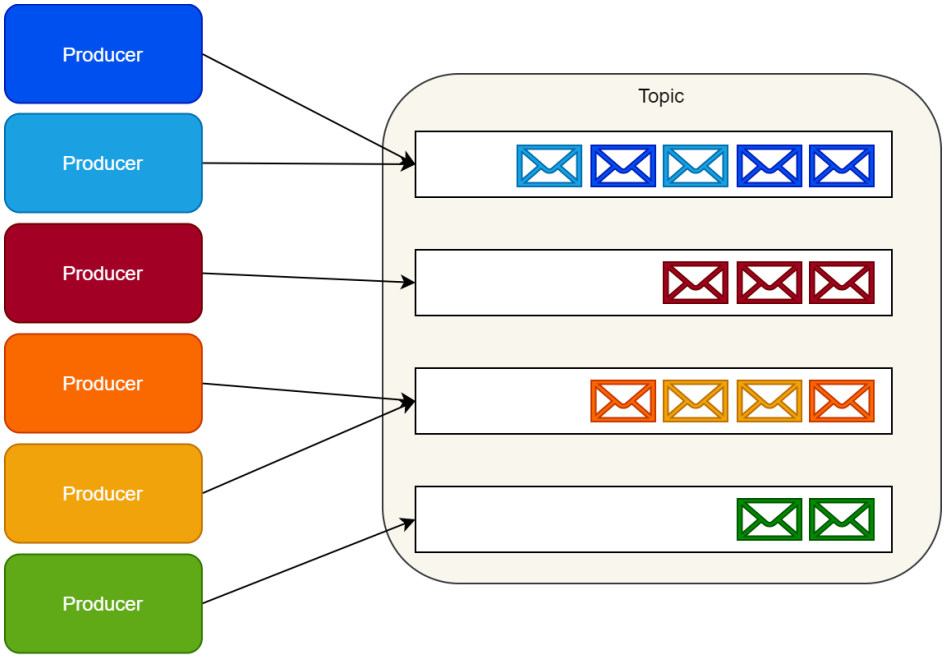


Рисунок 1.18 – Кафка видавці

Споживачі споживають повідомлення, підтримуючи зсув (або індекс) до цих розділів і читаючи їх послідовно.

Один споживач може споживати кілька тем, а споживачі можуть масштабувати до кількості доступних розділів.

В результаті, створюючи тему, слід ретельно продумати очікувану пропускну здатність обміну повідомленнями на цю тему.

Реалізація Кафки досить добре відповідає шаблону pub / sub. Виробник може надсилати повідомлення на певну тему, а декілька груп споживачів можуть споживати одне і те ж повідомлення. Кожна група споживачів може індивідуально масштабувати обробку навантаження. Оскільки споживачі підтримують зміщення розділів, вони можуть вибрати тривалу підписку, яка зберігає зміщення при перезапуску, або ефемерну підписку, яка відкидає зміщення та перезавантажує останні записи в кожному розділі кожного разу, коли він запускається.

Однак він менш ніж ідеально підходить для шаблону черг повідомлень. Звичайно, ми могли б мати тему лише з однією групою споживачів для наслідування класичної черги повідомлень.

Також важливо зазначити, що Kafka зберігає повідомлення в розділах до попередньо налаштованого періоду, незалежно від того,чи споживали ці повідомлення споживачі. Це збереження означає, що споживачі можуть вільно перечитувати минулі повідомлення. Крім того, розробники можуть також використовувати рівень зберігання Kafka для реалізації таких механізмів, як пошук подій та журнали аудиту.

Тож можна зробити висновок, що RabbitMQ є посередником повідомлень, тоді як ApacheKafka - розподіленою потоковою платформою. Ця різниця може здатися семантичною, але вона тягне за собою серйозні наслідки, які впливають на здатність комфортно реалізовувати різні випадки використання.

Наприклад, Kafka найкраще використовувати для обробки потоків даних, тоді як RabbitMQ має мінімальні гарантії щодо упорядкування повідомлень у

потоці. З іншого боку, RabbitMQ має вбудовану підтримку повторної логіки та обміну мертвими буквами, тоді як Kafka залишає такі реалізації в руках своїх користувачів.

RabbitMQ надає мало гарантій щодо впорядкування повідомлень, що відправляються в чергу або на біржу. Хоча може здатися очевидним, що споживачі обробляють повідомлення в тому порядку, в якому їх відправляли виробники, це дуже вводить в оману. Ця відсутність гарантії трапляється через те, що споживачі можуть повернути (або повторно доставити) повідомлення в чергу після їх прочитання (наприклад, у разі помилки обробки).

Після того, як повідомлення повертається, інший споживач може забрати його для обробки, навіть якщо він вже спожив пізніше повідомлення. Таким чином, групи споживачів обробляють повідомлення не у правильному порядку. Звичайно, можна було б уникнути цієї проблеми обмеживши кількість потоків в межах одного споживача повинна бути обмежена до одного, однак обмеження себе однопоточним споживачем суттєво впливає на здатність масштабувати обробку повідомлень у міру зростання системи. Натомість Kafka забезпечує надійну гарантію порядку на обробку повідомлень. Кафка розміщує повідомлення в розділах методом round-robin. Однаквиробникможевстановити ключ розділу для кожного повідомлення, щобстворитилогічні потоки даних (наприклад, повідомленнявід одного пристрою абоповідомлення, що належать одному клієнту).

Потімусіповідомлення з одного потоку розміщуються в одному розділі, в результатічогогруписпоживачівїхобробляють у правильному порядку.Однакслідзазначити,

щовсередінігруписпоживачівкоженрозділобробляється одним потоком одного споживача. Як результат, ми не можемомасштабуватиобробку одного розділу, однак у Kafka ми можемомасштабуватикількістьрозділів у межах теми, змушуючикоженрозділотримуватименшеповідомлень та додаватидодатковихспоживачів для додатковихрозділів.

RabbitMQможенаправлятиповідомлення абонентам обмінуповідомленнями на основівизначених абонентом правил маршрутизації. Обмін темами можеспрямовуватиповідомлення на основіспеціального заголовка. Крім того, обмін заголовками моженаправлятиповідомлення на основідовільнихзаголовківповідомлень. Кафка, навпаки, не дозволяєспоживачамфільтруватиповідомлення в темі перед їхопитуванням. Споживач, якийпідписався, отримуєвсіповідомлення у розділі без винятку. Для цьогоможнавикористовуватизавданняKafkaStreams, яке зчитуєповідомлення з теми, фільтруєїх і штовхає до іншої теми, на яку споживачможепідписатися. Тим не менше, цевимагаєбільшихзусиль та технічногообслуговування та маєбільшерухомихчастин.

RabbitMQпідтримуєвідкладені / заплановаріповідомлення за допомогоюплагіна. Коли цейплагінувімкненопід час обмінуповідомленнями, виробникможенадіслатиповідомленняRabbitMQ тазатримати час, протягомякогоRabbitMQнаправляєцеповідомлення в чергуспоживача.Цяфункціядозволяєрозробникупланувати майбутнікоманди, які не призначені для обробкираніше. Kafka не надаєпідтримки таких функцій. Ця системапишеповідомлення до розділів у мірунадходження, де вони одразустаютьдоступними для читанняспоживачами.

RabbitMQвидаляєповідомленнязісховища, як тількиспоживачіуспішноїхчитають. Цюповедінку не можназмінити. Цечастина дизайну майжевсіхпосередниківповідомлень.На відмінувідцього, Кафка зберігаєвсіповідомлення, тримаючиїх до встановленого часу очікування для кожної теми. Щостосуєтьсязбереженняповідомлень, Кафка не дбає про стан читаннясвоїхспоживачів, оскількивіндіє як журнал повідомлень.Споживачіможутьчитатикожнеповідомленняскількизавгодно, і вони можутьподорожувати«назад у часі», маніпулюючизміщеннямїхрозділів. Періодично Кафка переглядаєвікповідомлень у темах і видаляєтіповідомлення, які є старими.ЕфективністьроботиKafka не залежитьвідрозмірусховища. Отже,

теоретично можназберігатиповідомленнямайженескінченно, не впливаючи на продуктивність (якщовашівузлидоситьвеликі для зберіганняцихрозділів).

1.3.3 Поняття часу: час події та час обробки

Посилаючись на час у програмі потокового передавання (наприклад, для визначення вікон), можна посилатися на різні поняття часу.

Час обробки відноситься до системного часу машини, яка виконує відповідну операцію. Коли працює програма обробки потокових даних - усі операції, що базуються на часі (наприклад, часові вікна), використовуватимуть системний годинник машин, на яких запущений відповідний оператор. Вікно такої обробки включатиме всі записи, які надійшли до певного оператора між часами, коли системний годинник вказував повну годину. Час обробки є найпростішим поняттям часу і не вимагає координації між потоками та машинами. Це забезпечує найкращу продуктивність та найменшу затримку. Однак у розподіленому та асинхронному середовищі час обробки не забезпечує детермінованість, оскільки він сприйнятливий до швидкості, з якою записи надходять у систему (наприклад, з черги повідомлень), до швидкості, з якою записи протікають між операторами всередині системи , а також до відключень (запланованих чи інших).

Час події - це час, коли кожна окрема подія відбувалась на своєму виробничому пристрої. Цей час, як правило, вбудовується в записи перед тим, як вони потрапляють до фреймворку для обробки, і мітку часу події можна витягти з кожного запису. У цьому разі хід часу залежить від даних, а не від будь-яких годинників. Програми побудовані на основі часу подій повинні вказувати, як генерувати водяні знаки часу події, що є механізмом, який сигналізує про прогрес у часі подій.

У ідеальному світі обробка часу подій дасть цілком послідовні та детерміновані результати, незалежно від того, коли події надходять, або їх упорядкування. Однак, якщо події, як відомо, не надходять у порядку (за

позначкою часу), обробка часу подій спричиняє деяку затримку під час очікування випадків, що не працюють. Оскільки можна чекати лише кінцевий проміжок часу, це обмежує можливості детермінованих програм основаних на часу подій.

Якщо припустити, що всі дані надійшли, операції основані на часу подій працюватимуть належним чином і даватимуть правильні та послідовні результати, навіть під час роботи з невпорядкованими або пізніми подіями, або при обробці історичних даних. Наприклад, щогодинне вікно часу події міститиме всі записи, які містять позначку часу події, яка потрапляє в цю годину, незалежно від порядку, в якому вони надходять, або коли вони обробляються.

1.3.4 Базаданихчасовихрядів

База даних часових рядів - це база даних, оптимізована для даних із часовою міткою. Дані часових рядів – цевимірювання або події, які відстежуються, потрапляють до вибірки та агрегуються протягом часу. Це можуть бути метрики серверу, моніторинг продуктивності додатків, мережеві дані, дані датчиків, події, кліки, торгівлі на ринку та багато інших типів аналітичних даних.

База даних часових рядів створена спеціально для обробки метрик та подій або вимірювань, що мають часові позначки. Такі бази оптимізовані для вимірювання змін у часі. Властивості, які роблять дані часових рядів дуже відмінними від інших - це управління життєвим циклом даних, узагальнення та сканування великого діапазону багатьох записів.[11]

Бази даних часових рядів не є новими, але бази даних часового ряду першого покоління в основному були зосереджені на аналізі фінансових даних, волатильності біржової торгівлі та системах, побудованих для вирішення торгів. Але фінансові дані вже не є єдиним застосуванням даних часових рядів - насправді, це лише одне з численних застосувань у різних галузях. Основні

умови обчислювальної техніки різко змінилися за останнє десятиліття.Сьогодні все, що може бути компонентом, є компонентом.[11] Крім того, ми спостерігаємо інструментарій усіх доступних поверхонь у матеріальному світі - вулиць, автомобілів, фабрик, електромереж, крижаних шапок, супутників, одягу, телефонів, мікрохвильовок, контейнерів для молока, планет, людських тіл. Все має або буде мати датчик. Отже, все, що є в компанії та за її межами, випромінює невпинний потік метрик та подій або даних часових рядів.Це означає, що базові платформи повинні розвиватися для підтримки цих нових навантажень - більше точок даних, більше джерел даних, більше моніторингу, більше контролю.[12] Те, що ми спостерігаємо і чого вимагає час, - це парадигматичний зсув у тому, як потрібно підходити до інфраструктури даних і до побудови, моніторингу, контролю та управління системами. Потрібна продуктивна, масштабована, спеціально створена база даних часових рядів.

InfluxDB - це база данихчасовихрядів. Оптимізація для цьоговипадкувикористаннятягне за собою деякікомпроміси, насамперед для підвищенняпродуктивності за рахунокфункціональності. Нижче наведено перелікдеяких з деталей дизайну, якіпризводять до компромісів:

Увипадкувикористаннячасовихрядів, ми припускаємо, що якщоодні й тісамідданінадсилаютьсякількаразів, це абсолютно тісамідані, якіклієнтщойнонадсилавкількаразів.Спрощеневирішенняконфліктівпідвищуєпродуктивністьзапису, але унеможливлюєзберіганнядублікатівданих.

Обмеження доступу до видаленьдозволяєзбільшитипродуктивністьзапитів і записів.

Данічасовихрядів - цепереважноновідані, якініколи не оновлюються, щопідвищуєефективністьзапитів і записів.

Переважнабільшістьзаписівстосуєтьсяданихізнедавнімипозначками часу, і данідодаютьсяу порядку зростання за часом. InfluxDBоптимізована під такий сценарій.

1.4 Висновки до розділу

Системи обробки потокових даних мають великий набір доступних інструментів, кожен з яких має свої переваги та недоліки. Було розглянуто інструменти для аналізу даних, асинхроного обміну повідомленнями та збереження даних. Для описаних частин моєї системи найкраще підходять ApacheFlink, Kafka та InfluxDB.

Також у розділібулисформулюванізадачі, мета і призначеннядослідження.

.

2 АНАЛІЗ КОРИСНИХ МЕТРИК ДЛЯ ВИЯВЛЕННЯ ВІДХИЛЕНЬ

2.1 Двотактні двигуни внутрішнього згоряння

Для захисту дисертації я планую зосередитись на аналізу нормальної роботи саме двигунів морських суден. Головна причина, чому було обрана саме ця предметна область – доступність інформації. Також це досить цікава тема, для застосування набутих знань у повсякденому житті у використанні власного автомобіля.

На морських суднах зазвичай використовуються двотактні двигуни внутрішнього згоряння, що відрязняються від чотирьохтактних, які використовуються у автомобілях.

Як випливає з назви, двотактний двигун вимагає лише двох рухів поршня (один цикл), щоб виробляти потужність. Двигун здатний виробляти потужність після одного циклу, оскільки випуск і впуск газу відбувається одночасно. Існує клапан для впускного ходу, який відкривається і закривається внаслідок зміни тиску. Крім того, завдяки частому контакту з рухомими компонентами, паливо змішується з маслом для додавання мастила, що забезпечує більш плавні ходи.

Загалом двотактний двигун містить два процеси:

Такт стиснення: впускний отвір відкривається, повітряно-паливна суміш потрапляє в камеру і поршень рухається вгору, стискаючи цю суміш. Свічка запалює стиснене паливо і починає такт.

Такт сили: Нагрітий газ чинить сильний тиск на поршень, поршень рухається вниз (розширення), витрачене тепло відходить.

У порівнянні з чотиритактними двигунами, два такти легші, ефективніші, мають можливість використовувати менш якісне паливо та економічніші.Таким чином, легші двигуни призводять до вищого співвідношення потужності до ваги (більша потужність при меншій вазі). Однак їм не вистачає маневреності, можливої у чотиритактних двигунах, і вони потребують більше змащення. Це робить двотактні двигуни ідеальними для суден (потрібно перевозити багато

вантажу), мотоциклів та газонокосарок - тоді як чотиритактні ідеально підходять для автомобілів, таких як легкові та вантажні автомобілі.

Діаграма об'єму тиску (PV-діаграма), що моделює зміни тиску та об'єму паливо-повітряної суміші в будь-якому бензиновому двигуні, називається циклом Отто. Зміни в них будуть створювати тепло і використовувати це тепло для переміщення транспортного засобу або машини (отже, чому це тип теплового двигуна). Цикл Отто можна побачити на рисунку2.1 (реальний цикл Отто) та рисунку2.2 (ідеальний цикл Отто). Компонент будь-якого двигуна, що використовує цей цикл, матиме поршень для зміни об’єму та тиску паливно-повітряної суміші. Поршень отримує рух від спалювання палива та електричного підсилення при запуску двигуна.

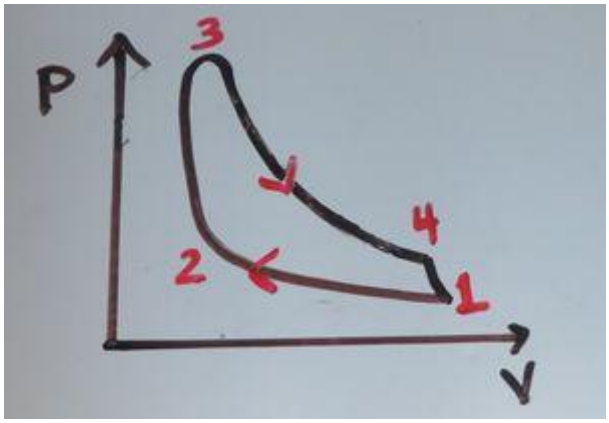


Рисунок 2.1 – Реальний цикл Отто

Ідеальна циклічно-зелена лінія: називається фазою впуску, але двотактний двигун не проходить цю фазу. Це пов’язано з тим, що чотиритактні двигуни починаються з підтягнутого поршня, тому його потрібно відтягувати для всмоктування паливно-повітряної суміші. Однак двотактний двигун може продовжувати всмоктування паливно-повітряної суміші відразу.

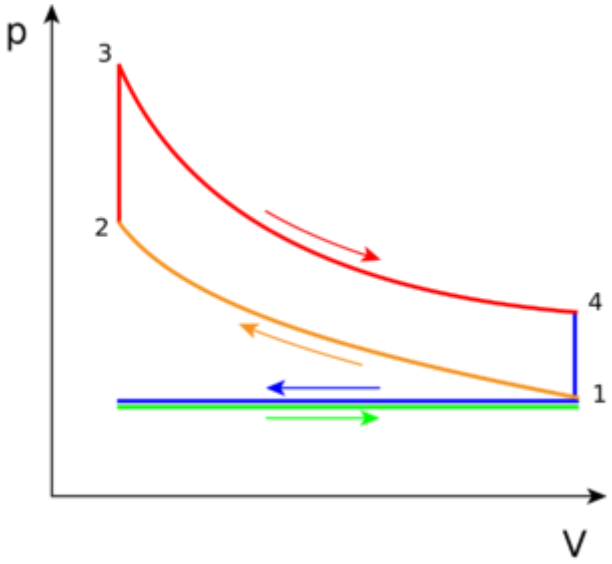


Рисунок 2.2 – Реальний цикл Отто

Фази роботи двотактного двигуна можна розбити на чотири процеси.

Процес 1-2: Під час цієїфазивпускнийотвірвідкривається, і поршень буде витягнутий, щобвінмігстиснутипаливно-повітрянусуміш, яка потрапила в камеру. Стисненняпризводить до незначногопідвищеннятискусуміші та температури. З точки зорутермодинамікиценазиваєтьсяадіабатичнимпроцесом. Коли цикл досягає точки 2 - паливодосягаєсвічкузапалювання.

Процес 2-3: Тут відбуваєтьсязгорянняявнаслідокзайманняпаливасвічкоюзапалювання. Горіння газу закінчується, в результатічогоутворюється камера з високимтиском, яка маєбагато тепла (тепловоїенергії). З точки зорутермодинамікиценазиваютьізохорнимпроцесом.

Процес 3-4: Теплова енергія в камері в результатігорінняявикористовується для роботи на поршні - якийштовхає поршень вниз - збільшуючиоб'ємкамери. Цетакожвідомо як запас енергії, оскількисаметодітепловаенергіяперетворюється в рух для живленнямашиниабо транспортного засобу.

Процес 4-1: З процесу 4 до 1 все відпрацьоване тепло виводиться з камеридвигуна. Коли тепло залишає газ, молекулявтрачаютькінетичнуенергію,

спричиняючизниженнятиску. Однак у двотактномудвигунінемаєвихлопноїфази, тому цикл починаєтьсязнову (від 1 до 2), дозволяючистискатиновусумішпалива та повітря.

2.1.1 Головні показники нормальної роботи двигуна

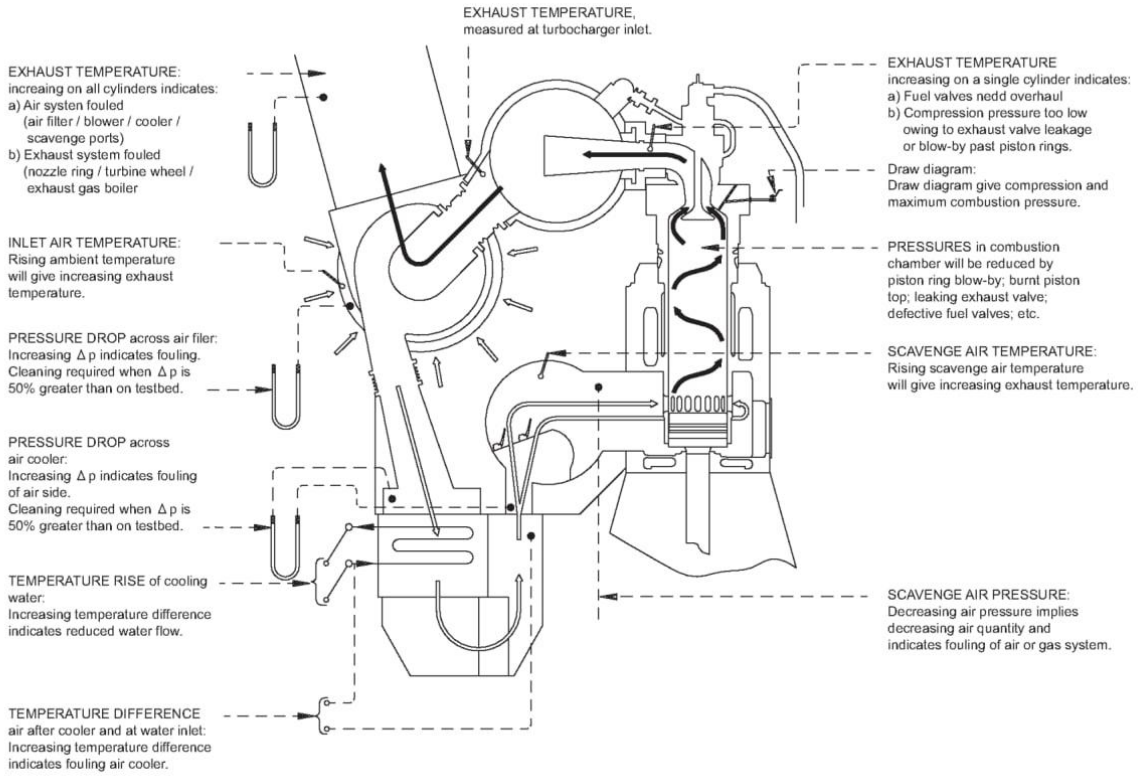


Рисунок 2.3 – Двигун внутрішнього згоряння

Температура вихлопних газів:

Збільшення на всіх циліндрах означає:

- повітряна система забруднена (повітряний фільтр / продувка/ охолоджувач/ порти для очищення);
- забруднена випускна система (кільце форсунки / колесо турбіни / котел відпрацьованих газів.

Температура вхідногоповітря: підвищеннятемпературинавколишньогосередовищапризведе до підвищеннятемпературивихлопу.

Падіннятиску в повітропроводівказує на забруднення. Потрібнеочищення, коли показник на 50% більше, ніж на дослідномустенді.

Падіннятиску на охолоджувачіповітрявказує на забрудненняповітряноїчастини. Потрібнеочищення, колизначення на 50% більше, ніж на дослідномустенді.

Підвищеннятемпературиохолоджуючої води свідчить про зменшенняспоживання води.

Різницятемпературиповітряпісляохолоджувача та на вході водивказує на забрудненняохолоджувачаповітря.

Збільшеннятемпературивихлопу на одному циліндрівказує на:

- потрібен капітальний ремонт паливнихклапанів;
- тиск стиснення занадто низький внаслідок витоку вихлопних клапанів або продування повз поршневих кілець.

Тиск в камері згоряння зменщиться за рахунок:

- продувки поршневого кільця;
- згорілий верх поршня;
- негерметичний випускний клапан;
- несправні паливні клапани;

Підвищеннятемпературиповітря, щовідходить, призведе до підвищеннятемпературивихлопу.

Зниженнятискуповітря, щовідходить,передбачаєзменшеннякількостіповітря і свідчить про забрудненняповітряноїчигазовоїсистеми.

2.1.2 Метрика пікового тиску у циліндрі

Вимірювання пікового тиску - це простий, швидкий і дешевий спосіб обслуговування двигунів.

Коли двигун внутрішнього згоряння втрачає потужність, вимірювання компресії (пікового тиску) часто є першим кроком у пошуку потенційних

проблем. Під час вимірювання циліндрів, тиск на кожному такті стиснення на індикаторі пікового тиску (манометрі) збільшується, поки не буде досягнуто або майже досягнуто максимуму, вказаного виробником двигуна.

Якщо виміряний тиск значно нижчий, ніж зазначено в технічних характеристиках двигуна, це може свідчити про знос. При збільшенні максимального тиску такожзбільшуються механічні напруження.

Розглянемо як можна будувати графік зміни пікового тиску у циліндрі двигуна. Для побудови метрик будемовикористовуватиінтервал у одну хвилину.

Спочатку агрегуємо дані з датчиків випускного клапан за хвилину, щоб побудувати цикли роботи двигуна.

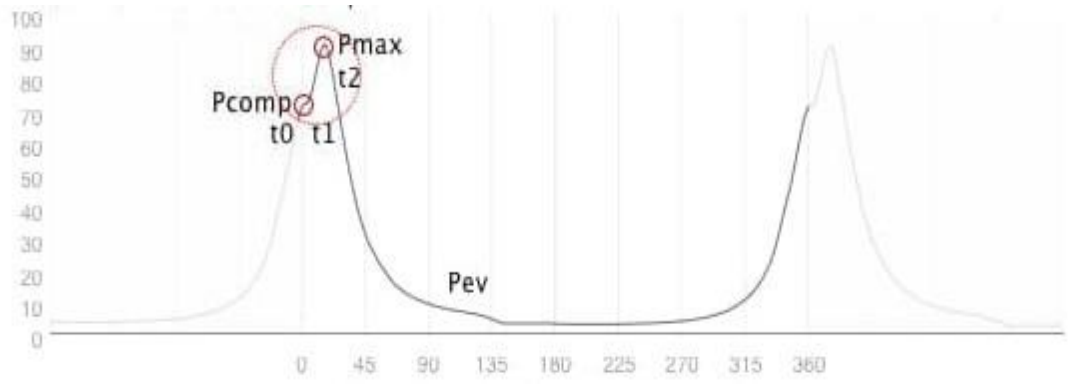


Рисунок 2.4 – Цикли роботи двигуна

Також потрібен другий потік даних, з датчику вимірювання тиску, який ми приєднуємо до потоку даних с циклами.

Тепер для кожного циклу ми маємо часовий ряд даних тиску, тож не важко знайти пікове значення тиску на першому діапазоні з валідованими даними.

Рисунок 2.5 – Алгоритм вимірювання пікового тиску

2.1.3 Вимірювання кількості обертів за хвилину (RPM)

Вимірювання кількості обертів колінчастого валу за хвилину також є дуже важливим показником двигуна.

При роботі двигуна на низьких обертах підвищується знос через те що навантаженні деталі змащуються недостатньо. Причиною є маслонасос. Його продуктивність і створюваний ним тиск моторного масла в мастильної системі залежить від обертів двигуна.

При високих обертах двигуна значно збільшується навантаження, що призводить до зносу.

Щоб порахувати цю метрику потрібно агрегувати дані з датчиків випускного клапану за хвилину та зсумувати кількість пікових значень.

2.2 Висновки до розділу

Другий розділ розглядає метрики роботи двигунів внутрішнього згоряння, які можна використовувати для моніторингу стану двигуна, та, якщо потрібно, планування зупинки на технічне обслуговування.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура системи потокової обробки на морському судні

Розглянемо архітектуру системи потокової обробки даних на морському судні.

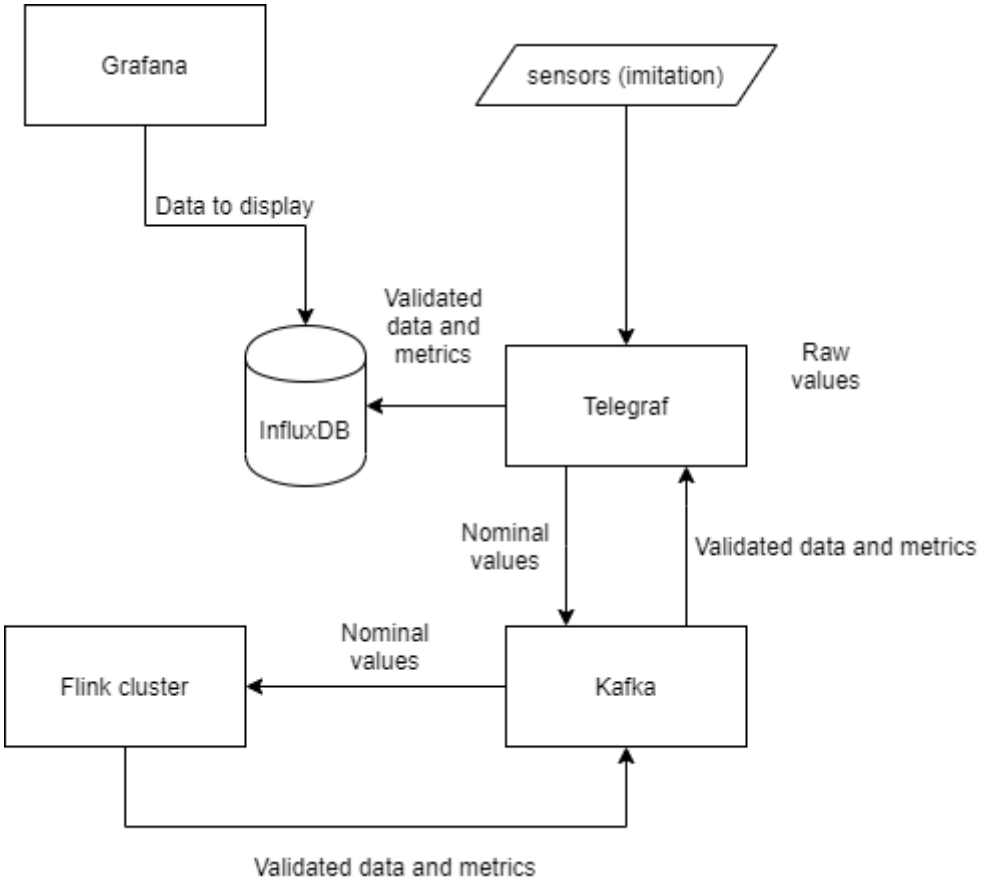


Рисунок 3.1 – Архітектура системи на морському судні

3.1.1 Опис рівнів системи на морському судні

Дані у систему потрапляють з рівня датчиків, які для магістерської ми будемо програмно імітувати.

Далі, використовуючи серверний агент Telegraf, ми забираємо дані з сенсорів та конвертуємо необроблені дані в номінальні дані і надсилаємо їх до рівня обміну повідомленнями.

З рівня обміну повідомленнями ми забираємо дані до рівня обробки, де працює кластер Флінку, обчислюємо та валідуємо метрики,повертаємо їх до

рівня обміну повідомленнями, де Телеграф забирає їх да надсилає до сховища даних часових рядів InfluxDB.

Для відображення даних на інтерфейсній частині використовується Grafana.

Для управліннясервісами використовується dockercompose.

Compose - цеінструмент для визначення та запуску багатоконтейнернихпрограмDocker. У composевикористовується файл YAML для налаштування служб системи. Потім за допомогоюоднієїкомандиможнастворитита запуститивсіслужби ззаданоїконфігурації.[16]

ВикористанняCompose - це в основному триетапнийпроцес:

- визначеннясередовищадодатка за допомогоюDockerfile, щобйогоможнабуловідтворювати де завгодно;
- визначення служб, з якихскладаєтьсядодаток,щобїхможнабулозапускати разом в ізольованомусередовищі;
- запуск docker-compose.

3.1.2 Таблиця класів сервісу для аналізу даних

Таблиця3.1 – Структура класів

Клас	Опис
DataVerification	Клас, що містить інформацію про оброблений дата фрейм, деталі валідації. Необхідний для побудови діаграми даних, пройшовшихвалідацію
FrequentDataValidatedMinute	Клас, що містить інформацію про пройшовшівалідацію дані з

	високою частотою, які було агреговані за хвилину.

Продовженнятаблиці3.1

Клас	Опис
FrequentDataPoint	Клас для представлення даних, які мають високу частоту
NonFrequentDataValidatedMinute	Клас, що містить інформацію про пройшовшівалідацію дані з низькою частотою, які було агреговані за хвилину. Необхідний для побудови діаграми даних, пройшовшихвалідацію
NonFrequentDataPoint	Клас для представлення даних, які мають низьку частоту
RangeInfo	Клас для представлення діапазону допустимих значень датчиків
DataMessage	Батьківський клас для даних з високою та низькою частотою, що містить спільні для них поля.
ConfugurationUploadException	Клас для представлення помилки читання конфігурації застосунку
MessageBrokerException	Клас для представлення помилки доступу до рівня обміну повідомленнями
InternalStateException	Клас для представлення помилки доступу до внутрішнього стану застосунку

Продовження таблиці 3.1

Клас	Опис
NonFrequentDataSelector	Функціональний клас для представлення методу вибору ключа для потоку даних з невисокою частотою
NonFrequentDataProcessor	Функціональний клас для представлення методу обробки агрегованих даних з невисокою частотою
NonFrequentDataFilter	Функціональний клас для представлення методу фільтрації агрегованих даних з невисокою частотою
FrequentDataSelector	Функціональний клас для представлення методу вибору ключа для потоку даних з високою частотою
FrequentDataProcessor	Функціональний клас для представлення методу обробки агрегованих даних з високою частотою
FrequentDataFilter	Функціональний клас для представлення методу фільтрації агрегованих даних з високою частотою
DataVerificationSerializer	Функціональний клас, що містить метод для серіалізації

Продовження таблиці 3.1

Клас	Опис
FrequentDataPointSerializer	Функціональний клас, що містить метод для серіалізації для об’єктів, що представляють дані з високою частотою
FrequentDataPointDeserializer	Функціональний клас, що містить метод для десеріалізації для об’єктів, що представляють дані з високою частотою
NonFrequentDataPointSerializer	Функціональний клас, що містить метод для серіалізації для об’єктів, що представляють дані з невисокою частотою
NonFrequentDataPointDeserializer	Функціональний клас, що містить метод для десеріалізації для об’єктів, що представляють дані з невисокою частотою
AppConfigUploader	Функціональний клас, що виконує завантаження налаштувань для застосунку
DataVerificationHelper	Допоміжний клас, що містить методи для валідації даних
NonFrequentDataPointTimestampExctractor	Функціональний клас, що обробляє час повідомлень з даними з невисокою частотою для просування часу у фреймворку обробки даних

Продовження таблиці 3.1

Клас	Опис
FrequentDataPointTimestampExctractor	Функціональний клас, що обробляє час повідомлень з даними з високою частотою для просування часу у фреймворку обробки даних
LoggerCreator	Клас для налаштувань логування
MessageBrokerAccessHelper	Допоміжний клас, що містить методи для роботи с брокером повідомлень
PeakPressureOutputMessage	Клас, що містить піковий тиск на часовому ряді агрегованих даних
PeakPressureOutputMessageSerializer	Функціональний клас, що містить метод для серіалізації для об’єктів, що представляють дані піковим тиском
RevolutionPerMinuteOutputMessage	Клас, що містить кількість обертів карданного валу двигуна за хвилину на часовому ряді агрегованих даних.
RevolutionPerMinuteOutputMessageSerial	Функціональний клас, що містить метод для серіалізації для об’єктів, що представляють кількість обертів карданного валу двигуна за хвилину

3.2 Повнаархітектурасистемиобробкипотоковихданих

У системі представлена також хмарна частина програмного забезпеченння, для використання даних з усіх суден. Розглянемо повну архітектуру, включаючи хмарну частину.

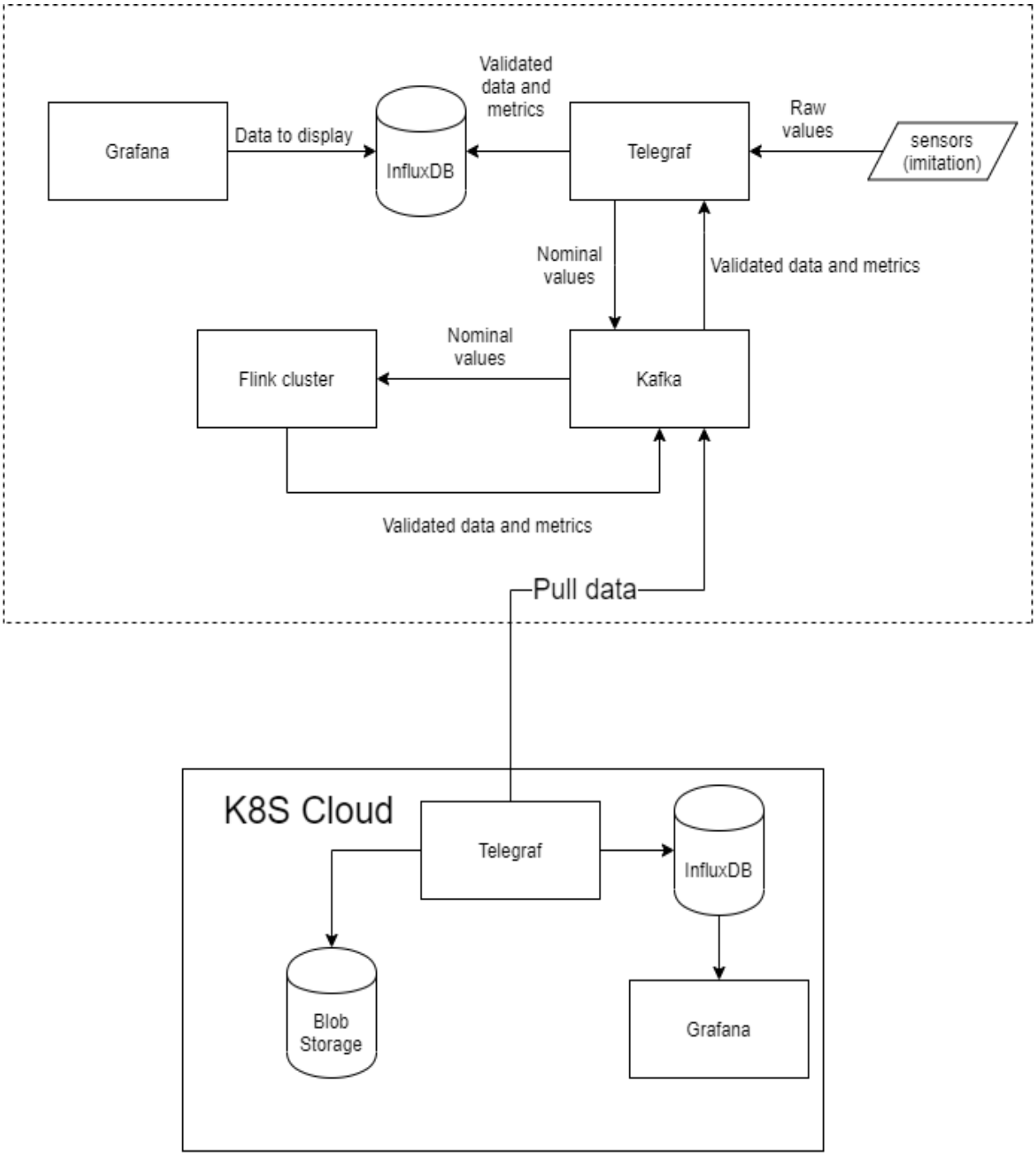


Рисунок 3.2 – Повна архітектура системи обробки потокових даних

Хмарна частина має власний інстанс телеграфу, сховища часових радів та графани. Також дані надсилаються до довговічного сховища даних, щоб використовувати їх у майбутньому.

Для організації сервісів у хмарному середовищі використовується Kubernetes.

Kubernetes - це платформа для організації контейнерів із відкритим кодом, яка автоматизує багато ручних процесів, що беруть участь у розгортанні, керуванні та масштабуванні контейнерних програм.[13]

Іншими словами, можна об’єднати групи хостів, на яких запущені контейнерилінукс, а Kubernetes допомагає легко та ефективно керувати цими кластерами.

Кластери Kubernetes можуть охоплювати хости на локальних, загальнодоступних, приватних або гібридних хмарах. З цієї причини Kubernetes є ідеальною платформою для розміщення власних хмарних додатків, які вимагають швидкого масштабування, наприклад потокового передавання даних у реальному часі через кафку.[13]

Основною перевагою використання Kubernetes у середовищі є те, що це дає платформу для планування та запуску контейнерів на кластерах фізичних або віртуальних машин.

Більш широко, це допомагає повністю впровадити та покластися на інфраструктуру на основі контейнерів у виробничих середовищах. І оскільки Kubernetes - це про автоматизацію операційних завдань, можна робити те саме, що дозволяють інші платформи програм або системи управління.

Розробники можуть також створювати власні хмарні програми з Kubernetes як платформою виконання, використовуючи шаблони Kubernetes. Шаблони - це інструменти, необхідні розробнику Kubernetes для створення додатків та служб на основі контейнерів.

3.3 Робота з інтерфейсом додатку

Щоб відкрити інтерфейс додатку потрібно перейти на сторінку веб-застосунку Grafana, який працює на порті 80.

Далі розглянемо метрики, доступні у застосунку, після проходженя системи потокової обробки даних.

Рисунок 3.3 — Графік кількості обертів за хвилину

Рисунок 3.4 — Графік пікового тиску у циліндрах

Для простоти сприйнятя, можна виділити лише один циліндр на графіку.

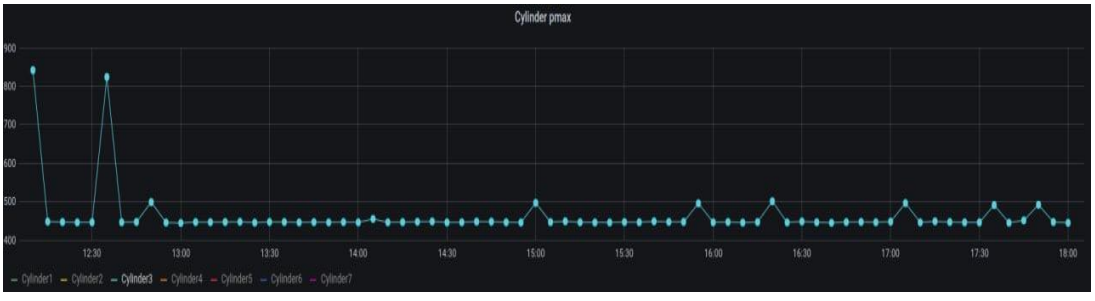


Рисунок 3.5 — Графік пікового тиску на одному циліндрі

Рисунок 3.6 — Показники турбіни, компресору та охолоджувача повітря

Рисунок 3.7 — Показники випускних клапанів циліндрів

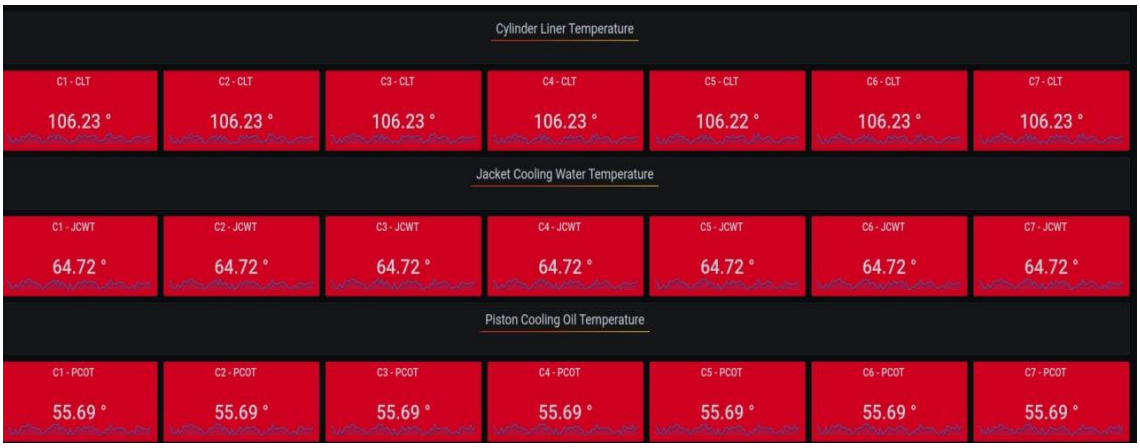


Рисунок 3.8 — Температурні показники ділянки охолодження

Як вже було описано вище, дані проходять процес валідації. Додатково прораховуються метрики доступності даних (чи було отримані всі дані за певний період часу) та чи всі дані мають адекватні значення (перевірка справності датчиків).

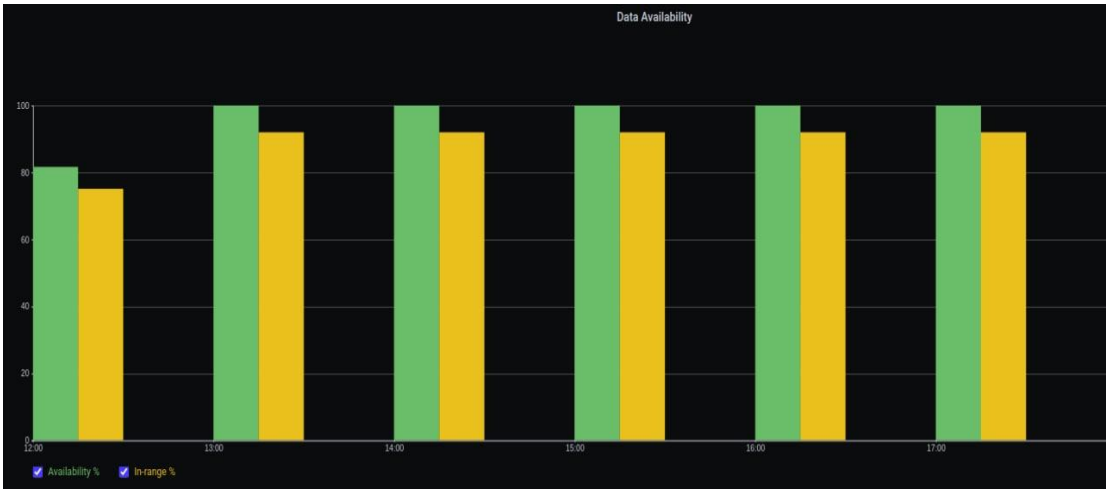


Рисунок 3.9 — Графік задовільних даних

Додатково також збирається інформація про сервер, на якому працює застосунок, що також є дуже важливою інформацією.

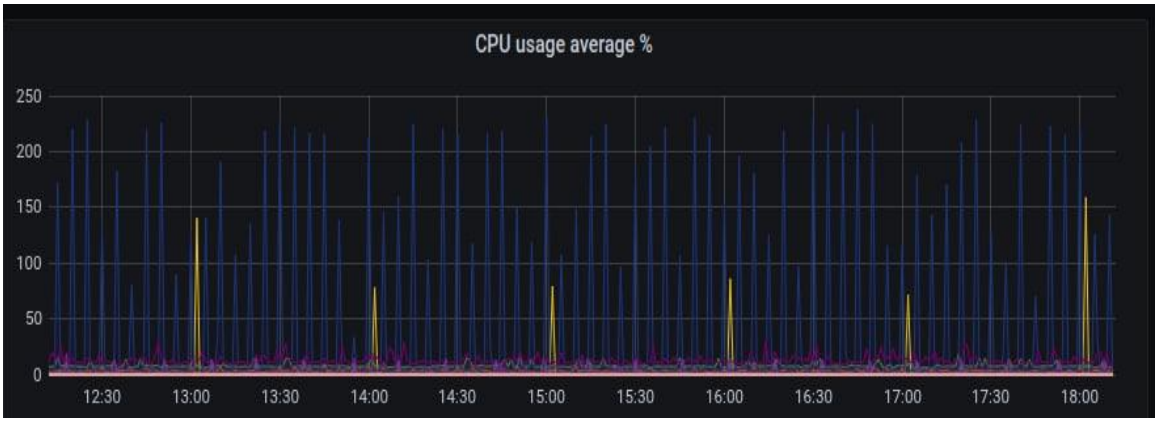


Рисунок 3.10 — Використанняпроцесору різними сервісами системи

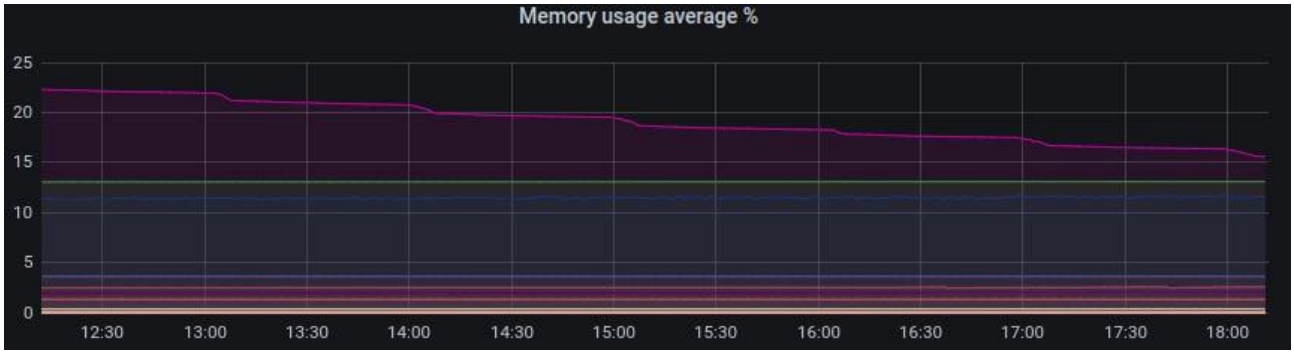


Рисунок 3.11 — Використанняоперативної пам’яті різними сервісами системи

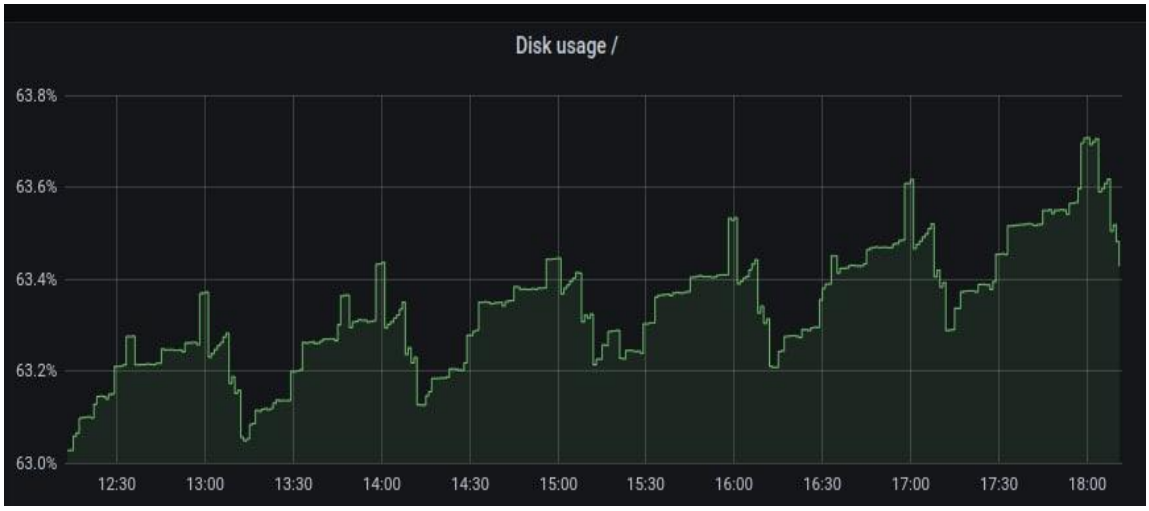


Рисунок 3.12 — Використанняжорстого диску різними сервісами

3.4 Висновки до розділу

У третьому розділі було розглянуто процес розробки системи обробки потокових даних, зокрема архітектуру системи на морських судах та у хмарі.

Також наведена таблиця класів застосунку та приклади веб-застосування Графана для відображення часових рядів з метриками, які відіграють важливо роль у моніторингу нормальної роботи суден.

4 РОЗРОБКА БІЗНЕС-ПЛАНУ ПРОЕКТУ

4.1 Опис ідеї проекту

Розглянемо зміст ідеї системи, напрямки її застосування та вигоду для користувача

Таблиця 4.1 –Ідея системи обробки потокових даних

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Розробити систему, що буде відслідковувати зміни у роботіморських суден	Дослідження стану морських суден під час їх пересування	Можливість моніторингу стану систем на морському судні під час його роботи
		Уникнення надлишкових зупинок для технічного обслуговування судна та виявлення відхилень
		Можливість використовувати машине навчання та історичні дані для передбачення пошкоджень

Необхідно порівняти ідею з уже існуючими реалізаціями та виявити переваги нової системи, тобто сильні сторони (S), нейтральні характеристики (N) та слабкі сторони(W)

Таблиця4.2 –Порівняння систем моніторингу даних

№	Функціонал	Сестеми конкурентів			W	N	S
		Проект описаний у дисертації	MarineTracker	ShipKeeper			
1	Збір метрик з різних робочих систем морського судна	Реалізовано тільки метрики роботи двигуна, але досить легко додати нові, інтегруючи їх в існуючу архітектуру системи	Реалізовано більше метрик, зокрема дані з навігаційного мосту, щогли, кормової частини судна	Реалізовані тільки метрики розташування суден		+	
2	Побудова метрик у реальному часі	Реалізовано за рахунок використання потокових даних, що оброблюються без затримки	Система використовую пакетну обробку, тому дані надходять із затримкою	Система використовую пакетну обробку, тому дані надходять із затримкою			+
3	Передбачення потенційних ушкоджень частин морського судна	Відсутнє, але має значний потенціал до реалізації за допомогою сховищ часових рядів	Відсутнє	Відсутнє			+

Продовженнятаблиці 4.2

		або побудови моделей для машиного навчання з використанням історичних даних					
4	Операційна система для виконання	На будь-якій операційній системі з використанням docker	Linux	Windows			+

Як бачимо, розроблена система має значні переваги, завдяки використанню новітніх технологій для обробки потокових даних. З недоліків можна зазначити, що система виконує моніторинг лише роботи двигуна, але архітектура відкрита до будь-яких нових частин суден.

4.2 Технологічний аудит ідеї проекту

Проведемо аналіз підсумкових інструментів, варіанти яких було описані у розділі з вимогами до програмного забезпечення. Завдяки ретельному дослідженню більшості технологій для різних рівнів системи можна було переконатись у роботоспроможності та надійності обраних технологій, що зможуть задовольнити усі функціональні вимоги до системи. Усі технології також мають досить великі товариства користувачів у інтернеті, тож не буде важко знайти підтримку у разі виникнення запитань чи проблем у процесі розробки.

Таблиця 4.3 – Технологічназдійсненністьсистеми обробки потокових даних

Ідея проекту	Технології реалізації	Наявність технології	Доступність технології
Система, що відслідковує зміни у нормальнійроботіморських суден	Датчики для вимірювання станів двигуна	Технологія наявна	Технологія доступна
	Фреймворк Flinkдля потокової обробки даних	Технологія наявна	Технологія доступна
	Алгоритми для підрахунку метрик	Розроблені у ході роботи над системою	Розроблені у ході роботи над системою
	Java Development Kit	Технологія наявна	Технологія доступна
	Telegraf	Технологія наявна	Технологія доступна
	Kubernetes	Технологія наявна	Технологія доступна
	InfluxDB	Технологія наявна	Технологія доступна
	Docker Compose	Технологія наявна	Технологія доступна
	Apache Kafka	Технологія наявна	Технологія доступна
	Apache Flink	Технологія наявна	Технологія доступна
	Grafana	Технологія	Технологія

		наявна	доступна

Тож у підсумку маємо такі інструменти для імплементації системи: датчики для вимірювання станів двигуна, фреймворк Flink, алгоритми для підрахунку метрик, javadevelopmentkit, telegraf, Kubernetes, influxdb, dockercompose, apachekafka, apacheflink, Grafana.

4.3 Аналіз ринкових можливостей запуску системи

Проаналізуємо ринок систем для моніторингу роботи морських суден.

Таблиця 4.4 – Характеристикапотенційногоринку системи моніторингу стану роботи морських суден

№	Показники стану ринку	Характеристика
1	Кількість головних гравців	Десятки
2	Динаміка ринку	Ринок росте з появою нових технологій
3	Наявність обмежень	Важко знайти тестові середовища для запуску
4	Специфічні вимоги до стандартизації	Відсутні
5	Середня норма рентабельності в галузі	~20%

З аналізу можна зробити висновок, що ринок є досить привабливим, з високою нормою рентабельності. Обмеження стосуються досить важкого отримання доступу до безпосередньо морських суден, для встановлення датчиків та серверів, які будут збирати та аналізувати інформацію.

Варто зазначити, що рентабельність залежить від кількості клієнтів, з якими вдасться підписати контракти.

4.4 Маркетингова програма стартап-проекту

Необхідно провести моделювання маркетингової програми стартап проекту.Для вдалихстосунківізсерьйознимиклієнтами система буде надаватисьбезкоштовно на тестовийперіод, щоклієнт мав змогупереконатись у дієздатнотісистеми та у тому, що вона реально допомогаєскоротитивитрати на технічнеобслуговування суден. Такий спосіб дозволяє отримати багато клієнтів на старті та прорекламувати продукт, готуючи клієнтів до контракту.

Проаналізуємопереваги використання системи, а також вигідні сторон, які отримує кінцевий користувач системи разом з перевагами використання системиствореної в дисертації.

Таблиця4.5 – Принципи використання продукту

Потреба	Вигідність при використанні	Переваги
Моніторинг стану морських суден під час їх роботи	Запобіганнянадлишковихзупинок для технічногообслуговування та економіявитрат	Побудовавихіднихданих у реальному часізавдякиновимтехнологіямBig Data

4.5 Ринкова стратегія системи

Проаналізуємопотенційнихкористувачівсистеми

Таблиця4.6 – Аналіз потенційних користувачів

Група	Готовність до користування	Попит	Конкуренція	Складність входу
Інженери на морських суднах, що можуть оцінити переваги системи	Готові	Високий	Висока	Середня
Власники логістичного бізнесу з використянням морських суден	Готові	Середній	Висока	Висока

Згідноз таблицею, стратегія продукту маєорієнтуватись на 2 групипотенційнихкористувачів, при чому вони маютьрізніпідходи до впровадження через різніможливості.Варто зазначити, що усе одно, навіть при орієнтації на інженерів – наша цільвласникибізнесу.

Для першоїгрупи, інженерівморських суден, більшпріоритетнітехнічніможливостісистеми.

Для другоїгрупиможнаорієнтуватись на рекламу, задлязавоюванняуваги та налаштуванняконтактів.

Такожнеобхідно обрати стратегію, яку стартап буде використовувати для виходу на ринок систем потоковоїобробкиінформації, зокрема для морських суден, задляуспішного запуску бізнесу і баченняйогорозвитку у найближчомумайбутньому.

Таблиця 4.7 – Стратегії виходу на ринок

Альтернатива розвитку	Стратегіяохоплення	Конкурентноспроможніпозиціїальтернативи	Основнийспосіброзвитку
Спеціалізація	Розвитоксистеми та реклама	Швидкість та надійністьроботи	Диференціація

За основу беретьсястратегіядиференціації — тобтоорієнтація на користувача. Якщоцейпідхід не спрацює буде застосованаспеціалізація для конкретноїгрупи людей.

Проаналізуємостратегіюконкурентноїповедінки на ринку.

Таблиця 4.8 – Конкурентна поведінка

Чи є проект першопрохідцем?	Шукати нових клієнтів чи забирати існуючих?	Чи буде копіюватиіснуючі характеристики товару конкурента?	Стратегія конкурентної поведінки
Ні	Завойовуватиіснуючих та шукатиклієнтів без систем	Ні	Зайняттяконкурентної ніші

Фінільна стратегія- зайняття ніші з охопленням кількохгруп споживачів.Продукт має аналоги на ринку, але все щеможназнайтиклієнтів, котрі не використовуютьсистемимоніторингу для роботисвоїх суден.

Проаналізуємостратегіюпозиціювання стартапу.

Таблиця4.9 - Позиціювання

Вимогивідцільової аудиторії	Базовастратегії розвитку	Ключові конкурентноспроможні показники	Вибірасоціацій, якімаютьсформуватикомплексупозиціювласного проекту
Точні метрики та надійнаобробкаінформації для швидкогоприйнятярішень на основівихіднихданих	Диференціація	Аналізданих у реальному часі	Якість та швидкість

4.6 Висновки до розділу

У четвертому розділі система була ретельно проаналізована для стартап проекту. Можна зробити висновок, що ринок досить активний та чекає нових гравців, через розвиток BigDatatехнологій.

Знайденодвігрупиклієнтів, наякінаціленийпродукт, проте основна група це власникилогістичного бізнесу. Середпереваг системибула виділена швидкий доступ до проаналізованих інсайтів. Тож можна зробити висновок, що проект має гарні перспективи на розвиток.

ВИСНОВКИ

У магістерській дисертації було виконане дослідження сьогоденних викликів світу інтернету речей та поставлена прикладна задача, для досягнення якої потребувався новий архітектурний підхід та використання сучасних технологій з домену великих даних.

Був виконаний аналіз існуючих інструментів для обробки потокових даних, змодельована та розглянута архітектура систем обробки потокових даних з урахуванням можливих складностей та потенційних проблем.

Для реалізації системи була побудована архітектура, розбита на окремі сервіси, кожен з яких відповідає за окремі задачі. Архітектура містить як серверну частину, так і хмарну.

Була проаналізована теорія роботи морських суден, зокрема морських двигунів. Виявлені метрики, які дають змогу оцінювати правильність роботи двигунів внутрішнього згоряння.

На основі виконаного аналізу була побудована система обробки потокових даних, яка допомагає власникам логістичного бізнесу з використанням морських суден заощаджувати кошти на технічному обслуговувані суден, уникаючи надлишкових зупинок.

Також було виконано розробку бізнес-плану проекту для запуску стартапу.

За результатами дисертації було опубліковано 1 публікацію на конференцію.

ПЕРЕЛІК ПОСИЛАНЬ

1) WhatisIoT? Theinternetofthingsexplained [Електронний ресурс] / режим доступу до статті: https://www.networkworld.com/article/3207535/what-is-iot-the-internet-of-things-explained.html

2) Andrew G. Psaltis (2017) StreamingData: Understandingthereal-timepipeline. Manning, America, 195.

3) ApacheFlink — StatefulComputationsoverDataStreams [Електронний ресурс] / режим доступу до статті: https://flink.apache.org/

4) ApacheSparkwebsite[Електронний ресурс] / режим доступу до статті: https://spark.apache.org/

5) DivingintoApacheSparkStreaming'sExecutionModel[Електронний ресурс] / режим доступу до статті: https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html

6) ApacheStormdocumentation- [Електронний ресурс] / режим доступу до статті: https://storm.apache.org/

7) ApacheSamza A distributedstreamprocessingframework [Електронний ресурс] / режим доступу до статті: http://samza.apache.org/

8) KafkaStreamsTheeasiestwaytowritemission-criticalreal-timeapplicationsandmicroservices [Електронний ресурс] / режим доступу до статті: https://kafka.apache.org/documentation/streams/

9) ApacheKafkadocumentation [Електронний ресурс] / режим доступу до статті: https://kafka.apache.org/

10) Messagingthatjustworks — RabbitMQ [Електронний ресурс] / режим доступу до статті: https://www.rabbitmq.com/

11) WhyShould I Use a TimeSeriesDatabase? [Електронний ресурс] / режим доступу до статті: https://thenewstack.io/use-time-series-database/#:~:text=Time%20series%20databases%20balance%20the,a%20higher%20number%20of%20writes.

12) ActinTime. BuildonInfluxDB [Електронний ресурс] / режим доступу до статті: https://www.influxdata.com/

13) KubernetesDocumentation [Електронний ресурс] / режим доступу до статті: https://kubernetes.io/docs/home/

14) InternalCombustionEngine [Електронний ресурс] / режим доступу до статті: https://www.sciencedirect.com/topics/engineering/internal-combustion-engine

15) Two-StrokeEngines: DefiningTheirPurpose[Електронний ресурс] / режим доступу до статті: https://www.cycleworld.com/2015/04/06/two-stroke-motorcycle-engines-explained-tech-talk-by-kevin-cameron/

16) OverviewofDockerCompose [Електронний ресурс] / режим доступу до статті: https://docs.docker.com/compose/

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ

DataVerification.java

```
publicclassDataVerification {



privateStringid;

privatelongtime;

privateStringship;



privatebooleanhasData;



privatebooleanisValid;



publicDataVerification() {}



publicDataVerification(

Stringid,

longtime,

Stringship,

booleanhasData,

booleanisValid) {



    this.id = id;

this.time = time;

this.ship = ship;

this.hasData = hasData;

this.isValid = isValid;

  }



publicStringgetid() {
```

```
returnid;

  }



publicvoidsetid(Stringid) {

    this.id = id;

  }



publiclonggettime() {

returntime;

  }



publicvoidsettime(longtime) {

this.time = time;

  }



publicStringgetship() {

returnship;

  }



publicvoidsetship(Stringship) {

this.ship = ship;

  }



publicbooleanishasData() {

returnhasData;

  }



publicvoidsethasData(booleanhasData) {
```

```
this.hasData = hasData;

  }



publicbooleanisisValid() {

returnisValid;

  }



publicvoidsetisValid(booleanisValid) {

this.isValid = isValid;

  }

}
```

FrequentDataValidatedMinute.java

```
publicclassFrequentDataValidatedMinute {



privateList<DataVerification>windowsDataVerification;



privateList<FrequentData>frequentDataList;



publicFrequentDataValidatedMinute() {}



publicFrequentDataValidatedMinute(List<DataVerification>windowsDataVerification) {

this.windowsDataVerification = windowsDataVerification;

  }



publicList<DataVerification>getWindowsDataVerification() {

returnwindowsDataVerification;

  }
```

```
publicvoidsetWindowsDataVerification(List<DataVerification>windowsDataVerification) {

this.windowsDataVerification = windowsDataVerification;

  }



publicList<FrequentData>getFrequentDataList() {

returnFrequentDataList;

  }



publicvoidsetFrequentDataList(List<FrequentData>FrequentDataList) {

this.FrequentDataList = FrequentDataList;

  }

}
```

FrequentDataPoint.java

```
publicclassFrequentDataPointextendsDataMessage {



privateStringvalue;



privateStringdimension;



publicFrequentDataPoint() {}



publicFrequentDataPoint(

longtimestamp,

Stringid,

Stringship,

Stringvalue,

Stringdimension,

Stringstatus) {
```

```
super(timestamp, id, ship, status);



this.value = value;

this.dimension = dimension;

  }



publicFrequentDataPoint(Stringstatus) {

super(status);

  }



publicStringgetvalue() {

returnvalue;

  }



publicvoidsetvalue(Stringvalue) {

this.value = value;

  }



publicStringgetdimension() {

returndimension;

  }



publicvoidsetdimension(Stringdimension) {

this.dimension = dimension;

  }

}
```

NonFrequentDataValidatedMinute.java

```
publicclassNonFrequentDataValidatedMinute {
```

```
privateList<DataVerification>windowsDataVerification;


privateNonFrequentDatanonFrequentData;


publicNonFrequentDataValidatedMinute() {}


publicNonFrequentDataValidatedMinute(List<DataVerification>windowsDataVerification) {
this.windowsDataVerification = windowsDataVerification;
  }


publicList<DataVerification>getWindowsDataVerification() {
returnwindowsDataVerification;
  }


publicvoidsetWindowsDataVerification(List<DataVerification>windowsDataVerification) {
this.windowsDataVerification = windowsDataVerification;
  }


publicNonFrequentDatagetNonFrequentData() {
returnNonFrequentData;
  }


publicvoidsetNonFrequentData(NonFrequentDataNonFrequentData) {
this.NonFrequentData = NonFrequentData;
  }
}
```

NonFrequentDataPoint.java

```
publicclassNonFrequentDataPointextendsDataMessage {


privatedoublevalue;


privateStringdimension;


privateStringmetric;


publicNonFrequentDataPoint() {}


publicNonFrequentDataPoint(Stringstatus) {
super(status);
  }


publicNonFrequentDataPoint(
longtimestamp,
Stringid,
Stringship,
Stringstatus,
doublevalue,
Stringdimension,
Stringmetric) {
super(timestamp, id, ship, status);
this.value = value;
this.dimension = dimension;
this.metric = metric;
  }
```

```
publicdoublegetvalue() {

returnvalue;

  }



publicvoidsetvalue(doublevalue) {

this.value = value;

  }



publicStringgetdimension() {

returndimension;

  }



publicvoidsetdimension(Stringdimension) {

this.dimension = dimension;

  }



publicStringgetmetric() {

returnmetric;

  }



publicvoidsetmetric(Stringmetric) {

this.metric = metric;

  }

}
```

RangeInfo.java

```
publicclassRangeInfo {



privatedoubleminimum;
```

```
privatedoublemaximum;



publicRangeInfo() {

    }



publicRangeInfo(doubleminimum, doublemaximum) {

this.minimum = minimum;

this.maximum = maximum;

    }



publicdoublegetminimum() {

returnminimum;

    }



publicvoidsetminimum(doubleminimum) {

this.minimum = minimum;

    }



publicdoublegetmaximum() {

returnmaximum;

    }



publicvoidsetmaximum(doublemaximum) {

this.maximum = maximum;

    }

}
```

DataMessage.java

```
publicclassDataMessage {


privatelongtime;


privateStringid;


privateStringship;


privateStringstatus;


publicDataMessage() {}


publicDataMessage(
longtime, Stringid, Stringship, Stringstatus) {


this.time = time;
    this.id = id;
this.ship = ship;
this.status = status;
  }


publicDataMessage(Stringstatus) {
this.status = status;
  }


publiclonggettime() {
returntime;
  }
```

```
publicvoidsettime(longtime) {

this.time = time;

  }



publicStringgetid() {

returnid;

  }



publicvoidsetid(Stringid) {

    this.id = id;

  }



publicStringgetship() {

returnship;

  }



publicvoidsetship(Stringship) {

this.ship = ship;

  }



publicStringgetstatus() {

returnstatus;

  }



publicvoidsetstatus(Stringstatus) {

this.status = status;

  }
```

```
}
```

ConfigurationUploadException.java

```
publicclassConfugurationUploadExceptionextendsRuntimeException {



publicConfugurationUploadException(StringerrorMessage, Throwableerr) {

super(errorMessage, err);

  }



publicConfugurationUploadException(StringerrorMessage) {

super(errorMessage);

  }

}
```

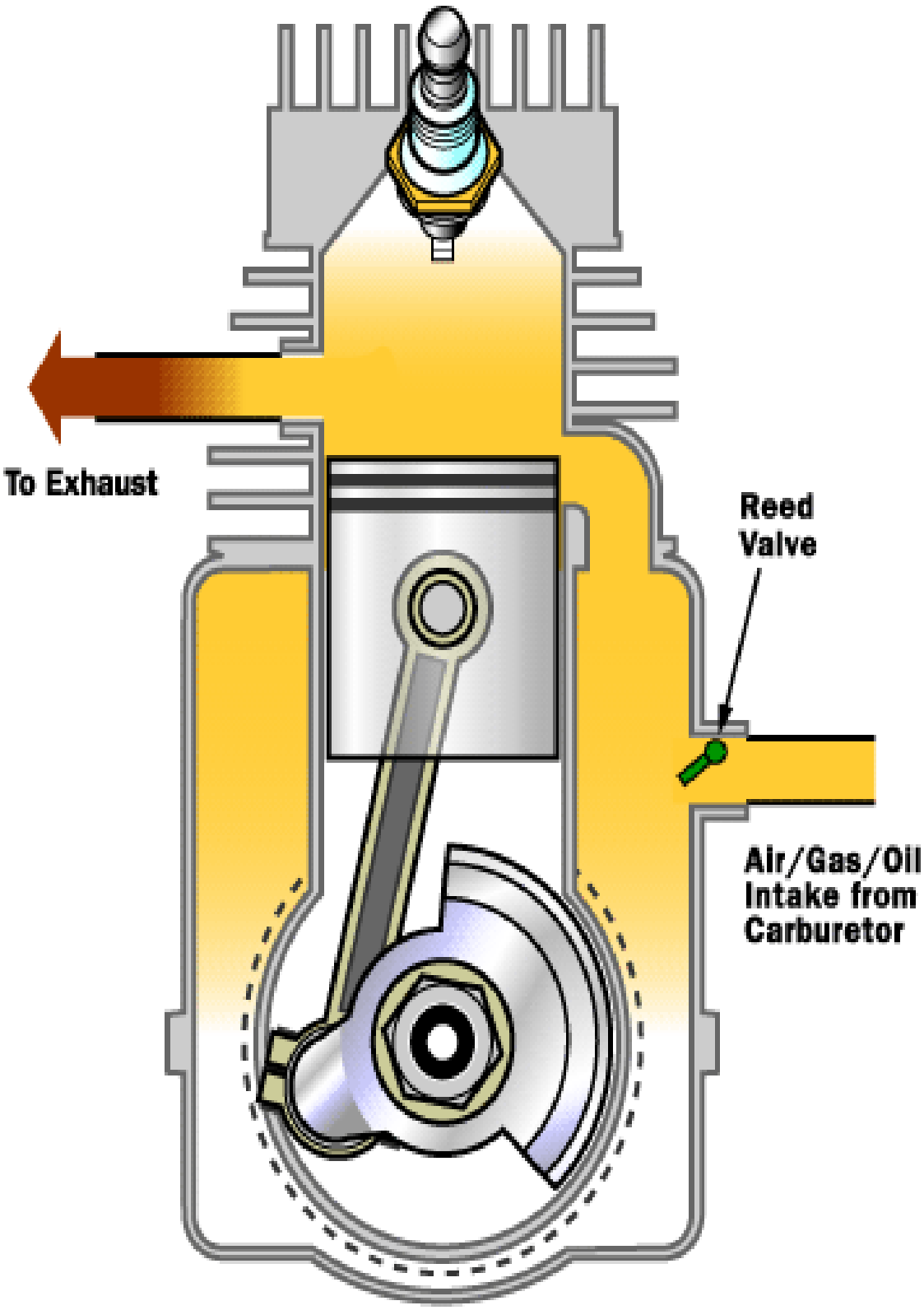
MessageBrokerException.java

```
publicclassMessageBrokerExceptionextendsRuntimeException {

publicMessageBrokerException(StringerrorMessage, Throwableerr) {

super(errorMessage, err);

  }



publicMessageBrokerException(StringerrorMessage) {

super(errorMessage);

  }

}
```

StateAccessException.java

```
publicclassStateAccessExceptionextendsRuntimeException {

publicStateAccessException(StringerrorMessage, Throwableerr) {

super(errorMessage, err);

  }

}
```

ДОДАТОК Б
СХЕМА ДВОТАКТНОГО ДВИГУНА ВНУТРІШНЬОГО ЗГОРЯННЯ


To Exhaust
Reed
Valve
Air/Gas/Oil
Intake from
Carburetor

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

ID перевірки:
1005466238

Дата перевірки:
16.12.2020 02:06:04 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.12.2020 02:25:21 EET

ID користувача:
77149

Назва документа: **Shulikov_magistr_ip92mp_2**

Кількість сторінок: **39** Кількість слів: **7046** Кількість символів: **55982** Розмір файлу: **93.93 KB** ID файлу: **1005755861**

3.96%
Схожість

Найбільша схожість: **1.19%** з джерелом з Бібліотеки (ID файлу: **1003048791**)

0.43% Джерела з Інтернету	15	Сторінка 41
3.96% Джерела з Бібліотеки	72	Сторінка 41

0.68% Цитат

Цитати	1	Сторінка 42

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел