Міністерство освіти і науки України
Львівський національний університет імені Івана Франка
Факультет електроніки і комп’ютерних технологій

Кафедра радіофізики та комп’ютерних технологій

КУРСОВА РОБОТА
на тему:
«Аналіз характеристик систем передачі та обробки повідомлень в розподілених системах»

Виконав:
Студент групи ФЕІМ-11С
Футей В.О.
Науковий керівник
доц. Вельгош С.Р.

Львів - 2023

Зміст

*Вступ* ..... *3*

*Kafka* ..... *5*

Партиції ..... 6

Практичне використання партицій ..... 7

Переваги ..... 8

Недоліки ..... 9

*Apache Hadoop* ..... *11*

Hadoop YARN ..... 12

Resource fragmentation ..... 13

*Apache Samza* ..... *16*

*Siddhi QL* ..... *18*

Розширення Siddhi QL ..... 18

Виявлення аномалій за допомогою Siddhi QL ..... 20

*Event processing* ..... *21*

*CEP (Complex Event Processing)* ..... *22*

Приклади ..... 22

Переваги ..... 24

Недоліки ..... 25

Альтернативні технології ..... 26

*Висновки* ..... *27*

*Література* ..... *28*

*Додаток* ..... *30*

Приклад додатку на Go що створює і відправляє повідомлення у Kafka: ..... 31

Приклад Samza додатку для підрахунку слів - high-level API: ..... 32

Приклад Samza додатку для підрахунку слів - low-level API: ..... 33

Приклад Samza додатку для підрахунку слів – з використанням Siddhi QL: ..... 34

# Вступ

Передача і обробка подій - це процеси, пов'язані зі збором, передачею, обробкою і реагуванням на різноманітні події або сповіщення, які відбуваються в системі або навколишньому середовищі.

Передача подій відбувається за допомогою різних механізмів, таких як протоколи зв'язку, передача даних через мережу або між компонентами системи. Події можуть бути сигналами, повідомленнями, сповіщеннями або даними, які передаються від одного компонента до іншого.

Після передачі подій вони підлягають обробці, що включає аналіз, інтерпретацію і виконання необхідних дій. Обробка подій може відбуватися на різних рівнях системи, включаючи апаратний рівень, операційну систему, програмне забезпечення або додатки.

Один із загальних підходів до обробки подій - це використання шаблону "підписка-публікація" (publish-subscribe), де компоненти системи можуть підписатися на певні типи подій і отримувати сповіщення, коли такі події виникають. Це дозволяє розділити виробників подій від їх споживачів і забезпечити гнучку комунікацію між ними.

Обробка подій може також включати додаткові дії, такі як логування, зберігання в базі даних, генерацію нових подій або сповіщень, аналіз даних і виконання бізнес-логіки системи.

Прикладами передачі та обробки подій є системи моніторингу, системи керування подіями, системи сповіщень або реагування на події в реальному часі, а також багато інших додатків, де важлива взаємодія між компонентами системи на основі подій.

В даній роботі ми розглянемо систему передачі і обробки подій базовану на існуючих компонентах, обговоримо підхід Complex Event Processing та його практичне застосування, обговоримо його сильні і слабкі сторони, коротко опишемо конкурентів. Також розглянемо можливості покращення системи в майбутньому.

Як систему передачі подій вибрано Apache Kafka (аргументи такого рішення подано далі у відповідному розділі), система обробки подій – Apache Samza

розширена із Siddhi QL. Також використано Apache Hadoop YARN як платформу для виконання додатків Samza (в майбутньому це дасть змогу досить легко реалізувати масштабованість).

## Kafka

*Apache Kafka отримав свою назву на честь письменника Франца Кафки. Назва Kafka була обрана з урахуванням його спадщини та впливу на літературу. Засновники проекту Apache Kafka вважали, що його архітектура та функціональність відображають багато з тих самих концепцій, які присутні в творчості Франца Кафки.*

*Кафка був відомий своїми роботами, які часто включали надзвичайно детальний аналіз та обробку інформації. Аналогічно, Apache Kafka спеціалізується на потоковій обробці даних, зберіганні та передачі повідомлень.*

*Таким чином, назва Kafka була вибрана, щоб віддати шану Францу Кафкі та підкреслити зв'язок між його творчістю і концепціями, які реалізовані у фреймворку Apache Kafka.*

Apache Kafka - це розподілена платформа обробки потокових даних та посередник (message broker), розроблена компанією LinkedIn та випущена як відкрите програмне забезпечення під егідою Apache Software Foundation.

Kafka була створена для вирішення проблем обміну потоковою інформацією в розподілених системах. Він дозволяє публікувати, підписуватись та обробляти великі обсяги потокових даних в режимі реального часу. Основними концепціями, на яких ґрунтується Kafka, є:

1. Повідомлення: Дані в Kafka представлені у вигляді повідомлень. Повідомлення можуть бути будь-якого формату або розміру, і вони організовані в теми (topics). Теми є категоріями або потоками даних, до яких можуть підписуватись споживачі.
2. Постачальники та споживачі: Постачальники (producers) публікують повідомлення в Kafka в певну тему, тоді як споживачі (consumers) підписуються на тему та отримують повідомлення з неї для обробки. Декілька споживачів можуть читати з того самого топіка незалежно – Kafka надає для цього стійкий до відмов механізм, що базується на збереженні зміщення (offset) для кожного споживача незалежно.

3. Брокери та кластери: Kafka працює у розподіленому середовищі, де повідомлення зберігаються на брокерах. Брокери Kafka утворюють кластер, де кожен брокер містить частину повідомлень та обслуговує певну кількість тем. Також за допомогою брокерів імплементована стійкість до відмов і масштабованість.
4. Потокова обробка: Однією з ключових функцій Kafka є здатність обробляти потоки даних у режимі реального часу. Завдяки потоковій обробці (stream processing) можна реалізовувати різноманітні операції, такі як агрегація, фільтрація, об'єднання тощо безпосередньо внутрішньо самої платформи Kafka.
5. Партиції: топіки у Kafka поділено на партиції, що можуть зберігатись на різних брокерах, завдяки чому досягається висока доступність (high availability) і паралельна обробка даних
6. Зміщення (Offsets): унікальний ідентифікатор прив’язаний до кожного повідомлення в партиції. Споживачі використовують зміщення, щоб відслідковувати свій прогрес обробки повідомлень і бути впевненими що вони опрацьовуються у правильному порядку.

Apache Kafka використовується у різних сферах, де важлива обробка потокових даних у реальному часі. Його застосовують для синхронізації даних між мікросервісами, збереження журналів, потокової обробки даних, аналітики в реальному часі та багатьох інших сценаріїв.

**Партиції**

У Apache Kafka, партіції є основною одиницею розподілення та збереження потоку даних. Кожна тема (topic) у Kafka може бути розбита на одну або більше партіцій. Кожна партіція представляє впорядковану послідовність записів (повідомлень), які зберігаються у ній.

Основні аспекти партіцій у Kafka:

1. Розподілення навантаження: Використання партіцій дозволяє розподілити навантаження між багатьма брокерами (Kafka brokers) у кластері. Кожен брокер може обробляти певну кількість партіцій.
2. Масштабованість: Додавання нових партіцій у тему дозволяє масштабувати обробку даних у Kafka. Кожна партіція може бути розподілена по різних брокерах, що дозволяє розпаралелювати обробку повідомлень.
3. Порядок повідомлень: Всередині кожної партіції повідомлення зберігаються у порядку їхнього надходження, тобто гарантується впорядкованість записів всередині партіції. Однак, порядок між різними партіціями та різними темами не гарантується.
4. Реплікація: Кожна партіція може мати декілька реплік (реплікаційних копій) для забезпечення надійності та відновлення даних в разі відмови брокера. Репліки розміщуються на різних брокерах, що дозволяє розподілити ризик втрати даних.
5. Консюмери та групи споживачів: Консюмери можуть споживати дані з однієї або декількох партіцій. Кожна група споживачів може мати свою власну зміщену позицію (offset) для кожної партіції, що дозволяє багатьом споживачам одночасно споживати дані з теми.

Використання партіцій у Kafka дозволяє ефективно розподіляти та обробляти великі обсяги даних, забезпечувати надійність та забезпечувати гнучкість у споживанні потоків даних.

**Практичне використання партицій**

У Kafka партіції використовуються для розподілу повідомлень в топіках між брокерами та забезпечення паралельної обробки даних. Практичне використання партіцій у Kafka включає:

1. Горизонтальне масштабування: Партіції дозволяють розподіляти великі обсяги даних по різних брокерах у кластері Kafka. Це дозволяє горизонтально масштабувати систему, збільшуючи пропускну здатність та оброблюючи більше повідомлень паралельно.

2. Розділення навантаження: Коли топік має багато партіцій, кожна партіція може обробляти окрему частину повідомлень. Це дозволяє розділити навантаження між кількома споживачами, що працюють паралельно та обробляють різні партіції. Це особливо корисно, коли вам потрібно обробляти великий обсяг даних швидко.
3. Збереження даних: Партіції в Kafka забезпечують розподілене збереження даних. Кожна партіція може мати свою власну репліку на різних брокерах, що забезпечує надійність та відновлення даних в разі відмови. Дані в партіції зберігаються в послідовному журналі, що дозволяє ефективно обробляти та зчитувати дані.
4. Забезпечення порядку повідомлень: Партіції в Kafka гарантують збереження порядку повідомлень в рамках кожної партіції. Це означає, що всі повідомлення, надіслані до однієї партіції, будуть зберігатись та читатись в тому ж порядку, в якому вони були надіслані. Це особливо важливо для додроботи даних та збереження послідовності операцій.
5. Балансування навантаження: Партіції дозволяють розподіляти навантаження між різними брокерами в кластері Kafka. Kafka автоматично здійснює балансування партіцій, переміщуючи їх між брокерами відповідно до змін в конфігурації або стану кластера. Це дозволяє рівномірно розподіляти навантаження та забезпечувати високу доступність даних.

Загалом, використання партіцій у Kafka дозволяє побудувати масштабовані та стійкі розподілені системи обробки повідомлень з великим обсягом даних. Це дозволяє розподіляти навантаження, забезпечувати високу пропускну здатність та забезпечувати надійність та відновлення даних у випадку відмов.

**Переваги**

Apache Kafka є потужною і популярною платформою для передачі та обробки подій у розподілених системах. Ось деякі причини, чому варто вибрати Apache Kafka:

1. Висока пропускна здатність: Kafka побудована з фокусом на високій швидкості та надійності. Вона може обробляти великі обсяги подій із високою швидкістю навіть при великій кількості виробників та споживачів.
2. Масштабованість: Kafka розроблена для масштабованих розподілених систем. Вона може працювати на кластері з багатьма брокерами Kafka та підтримує горизонтальне масштабування, що дозволяє збільшувати потужність обробки за потребою.
3. Толерантність до помилок: Kafka забезпечує високий рівень надійності і толерантності до помилок. Вона зберігає дані подій на диску, що гарантує відновлення при відмові системи і забезпечує можливість повторного споживання подій.
4. Розширюваність: Kafka можна інтегрувати з іншими інструментами та екосистемами. Вона підтримує різні протоколи зв'язку, такі як REST, TCP і AMQP, що робить її варіантом для інтеграції з різними системами.
5. Багатофункціональність: Kafka надає розширений набір функціональностей, таких як можливість розподіленого зберігання, розгалуження подій (topic), розсилка (publish-subscribe), затримка (event time), потік подій (event streaming), агрегація та багато іншого.
6. Широке співтовариство: Apache Kafka є проектом з відкритим вихідним кодом, що має велике співтовариство користувачів і розробників. Це означає, що є багато документації, підтримки та ресурсів, які можна використовувати при роботі з Kafka.
7. Загалом, Apache Kafka є потужним інструментом для передачі та обробки подій, який може задовольнити потреби великих розподілених систем, де швидкість, масштабованість та надійність є важливими факторами.

**Недоліки**

Хоча Apache Kafka є потужним і популярним інструментом для обробки подій, він також має деякі недоліки. Ось декілька з них:

1. Складність налаштування: Налаштування та конфігурація Apache Kafka може бути відносно складною завданням, особливо для новачків. Існує багато параметрів і параметрів конфігурації, які можуть впливати на продуктивність та надійність системи. Неправильне налаштування може призвести до проблем з продуктивністю або надійністю.
2. Складність управління: Керування та моніторинг кластера Apache Kafka також може бути викликом. Кластери Kafka вимагають управління брокерами, темами, розділами та реплікацією. Налаштування, моніторинг та масштабування кластера можуть бути складними задачами, особливо при масштабуванні до великих обсягів даних.
3. Відсутність вбудованої підтримки для обробки подій в реальному часі: Apache Kafka сам по собі є системою потокового введення-виведення, а не системою комплексної обробки подій (CEP). Хоча ви можете використовувати Kafka для потокової обробки даних, він не має вбудованої функціональності для складної обробки подій, такої як виявлення шаблонів, розпізнавання послідовностей або вирішення складних логічних виразів.
4. Збереження даних: Apache Kafka зберігає дані властивим для нього способом, використовуючи розділення на партиції та реплікацію. Однак, у випадку використання Kafka як основного механізму зберігання даних, він може бути обмеженим з точки зору функціональності пошуку, запитів та аналізу даних порівняно з іншими спеціалізованими системами зберігання даних.

Все ж таки, Apache Kafka є потужним і ефективним інструментом для обробки подій у багатьох сценаріях, і багато з недоліків можна зменшити або подолати за допомогою правильного налаштування, управління та додаткових розширень.

## Apache Hadoop

*Назва "Hadoop" походить від імені плюшового ведмедика, якого належало дитинству Дуга Катчіна (Doug Cutting), одного зі співзасновників проекту Apache Hadoop. Катчін створив Hadoop разом з Майклом Кафферіо (Mike Cafarella) з метою розробки системи для обробки та аналізу великих обсягів даних.*

*Початково проект називався Nutch, але назва була змінена на Hadoop після того, як Катчін почув, як його син називає його плюшового ведмедика "Hadoop". Ця назва стала прийнятою і була зафіксована для проекту.*

*Таким чином, назва Hadoop виникла з приватного дитячого імені плюшового ведмедика та була використана для проекту з обробки великих обсягів даних, який пізніше став відомим як Apache Hadoop.*

Apache Hadoop є відкритим фреймворком для обробки та збереження великих обсягів даних у розподілених середовищах. Він надає ефективні засоби для обробки, аналізу та збереження даних, які розподілені на кластері з вузлів.

Hadoop складається з двох основних компонентів:

1. Hadoop Distributed File System (HDFS): HDFS є розподіленою файловою системою, яка призначена для збереження великих обсягів даних на кластері. Вона розділяє дані на блоки та реплікує їх на різних вузлах кластера для забезпечення надійності та доступності.
2. Hadoop MapReduce: MapReduce є моделью програмування та виконання, яка дозволяє обробляти великі обсяги даних у паралельному та розподіленому середовищі. Вона базується на ідеї розбиття обчислювальної задачі на менші підзадачі (Map) та подальшому комбінуванні їх результатів (Reduce).

Hadoop дозволяє паралельно обробляти великі обсяги даних на кластері, розподіляючи обчислювальні та зберігаючі завдання між вузлами кластера. Він забезпечує високу доступність даних та відмовостійкість шляхом реплікації даних на різних вузлах.

Hadoop став популярним великими компаніями та організаціями, які працюють з великими обсягами даних, оскільки він дозволяє ефективно обробляти та

аналізувати такі дані. Він також має багато екосистемних компонентів та інструментів, таких як Apache Hive, Apache Pig, Apache Spark, які дозволяють розширити можливості обробки та аналізу даних у Hadoop.

## Hadoop YARN

Apache Hadoop YARN (Yet Another Resource Negotiator) є компонентом відкритої платформи Apache Hadoop, яка відповідає за керування ресурсами та планування завдань у розподіленій обчислювальній системі.

Hadoop YARN виступає в ролі ресурсного менеджера, який ефективно розподіляє обчислювальні ресурси (процесорний час, пам'ять, пропускна здатність мережі) між різними додатками, які працюють в Hadoop-середовищі. Він дозволяє паралельно виконувати багато різних завдань на вузлах кластера, використовуючи доступні ресурси.

Hadoop YARN розширює можливості Hadoop, який початково був розроблений для обробки великих обсягів даних за допомогою MapReduce. Замість того, щоб обмежуватись лише моделлю MapReduce, YARN надає загальну платформу для розподілених додатків різних типів, таких як Samza, Spark, Hive, Impala, Flink та інші. Він дозволяє запускати різні завдання обробки даних, аналізу, машинного навчання та інших обчислювальних завдань на кластері Hadoop.

Основна ідея Hadoop YARN полягає у відокремленні обробки даних від керування ресурсами. Він дозволяє розробникам додатків фокусуватись на логіці своїх завдань, використовуючи абстракцію робочих контейнерів та механізмів планування, які надає YARN.

Завдяки Hadoop YARN можливе ефективне використання обчислювальних ресурсів кластера, підтримка багатозадачності та масштабованість, що робить його важливим компонентом для розподіленого обчислення в середовищі Hadoop.

## Resource fragmentation

Resource fragmentation (фрагментація ресурсів) - це проблема, яка виникає у розподілених системах, коли ресурси системи (такі як пам'ять, процесорний час, мережевий пропуск, дисковий простір і т.д.) розділяються між різними компонентами системи або процесами, і в результаті виникають фрагменти ресурсів, які не можуть бути ефективно використані.

Ця проблема може мати наступні наслідки:

1. Втрата продуктивності: Фрагментація ресурсів може призводити до зменшення ефективності використання ресурсів, оскільки частка ресурсів може бути недоступною або підвищувати затримки у доступі до них. Це може призвести до зниження продуктивності системи та збільшення часу відгуку.
2. Обмеження масштабованості: Фрагментація ресурсів може ускладнити масштабування системи, оскільки неможливо однозначно виділити та управляти ресурсами для нових компонентів або процесів. Це може створювати перешкоди для зростання системи та розширення її місткості.
3. Недооптимізоване використання ресурсів: Фрагментація ресурсів може призводити до недооптимізованого використання ресурсів, оскільки фрагменти ресурсів можуть залишатися невикористаними або бути використані неоптимально. Це може призводити до зайвого споживання ресурсів та зниження ефективності системи.

Для подолання проблеми фрагментації ресурсів можуть бути використані різні стратегії, такі як управління розподіленими ресурсами, планування ресурсів, використання віртуалізації або контейнеризації, адаптивне розподілення ресурсів та інші. Оптимальний вибір стратегії залежить від конкретного випадку використання та характеристик системи.

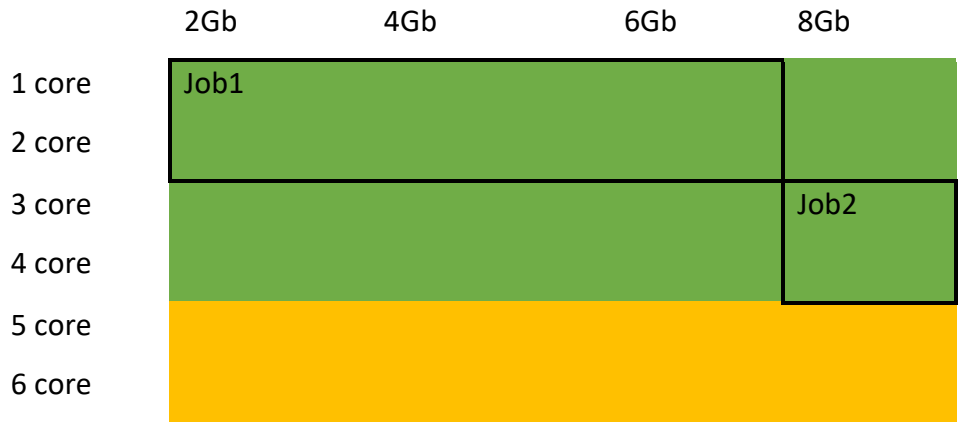
У нашому випадку ми будемо мати потенційні проблеми із фрагментацією по двох типах ресурсів: пам’ятю і процесорними ядрами. Проблема схожа із фрагментацією вільного дискового простору на накопичувачах, з тією різницею, що для дискового наокпичувача проблема існує в одновимірному просторі, а у нас – у двохвимірному, що ускладнює її вирішення. Частково уникнути проблеми

фрагментації по пам’яті можна зменшивши розмір контейнера для Samza додатку, проте для оптимальної його роботи це швидше за все потребує додати механізм масштабування по пам’яті. Інший варіант вирішення (причому по обох векторах) – збільшення потужностей нод, що працюють у складі Hadoop – але це рішення може бути не прийнятним внаслідок значного збільшення кошторису використання системи.

Нижче наведено приклади фрагментації ресурсів, нода що використовується на Hadoop має 8Gb пам’яті та 6 ядер.

| | Core(s) | Memory |
|---|---|---|
| **Job1** | 4 | 2 |
| **Job2** | 2 | 2 |
| | 6=6 | 4<8 |
| | All cores used | 4Gb unused |

Приклад невикористаної пам’яті (використанні всі процесорні ядра)




| | Core(s) | Memory |
|---|---|---|
| **Job1** | 2 | 6 |
| **Job2** | 2 | 2 |
| | 4<6 | 8=8 |
| | 2 core unused | All memory used |

Приклад невикористаних процесорний ядер (використанна вся пам’ять)

## Apache Samza

*Apache Samza була розроблена на основі досвіду з роботою над системою обробки потокових даних і була зорієнтована на розподілену обробку потокових даних у реальному часі. Розробка Apache Samza було започатковане командою інженерів в LinkedIn.*

Apache Samza є відкритою розподіленою платформою для обробки потокових даних, розробленою Apache Software Foundation. Вона дозволяє обробляти великі об'єми потокових даних з низькою затримкою та забезпечує стійкість до відмов. Samza спроектована для тісної інтеграції з Apache Kafka як системою передачі повідомлень, хоча може працювати і з іншими системами передачі повідомлень.

Основні особливості та концепції Apache Samza включають:

1. Обробка потоків: Samza дозволяє обробляти неперервні потоки даних в реальному часі. Вона надає простий API для розробки програм обробки потоків, надаючи абстракції для роботи з потоками даних, логікою обробки та управлінням станом.
2. Стійкість до відмов: Samza забезпечує стійкість до відмов завдяки лог-заснованому зберіганню та механізму реплікації Kafka. Вона гарантує, що дані не втрачаються навіть у разі відмов, зберігаючи вхідні та вихідні потоки в темах Kafka. Збереження стану програмного забезпечення досягається шляхом зберігання стану додатків у розподіленій системі зберігання даних, такій як Apache Kafka або Apache HBase.
3. Управління станом: Samza підтримує управління станом додатку, дозволяючи обробляти потоки даних та оновлювати стан по мірі надходження нових даних. Це корисно для завдань, таких як віконні операції, агрегації та обробки подій. Samza надає вбудовану підтримку для обробки зі збереженням стану, використовуючи розподілені системи зберігання, такі як Kafka або HBase.
4. Масштабованість: Samza спроектована для горизонтального масштабування, що дозволяє додаткам обробляти великі об'єми даних, розподіляючи робоче навантаження між кількома екземплярами. Вона автоматично розподіляє та

паралелізує обробку на основі кількості вхідних розділів Kafka та бажаного рівня паралелізму.

5. Гарантія обробки подій: Samza забезпечує точну обробку подій і гарантує, що кожна подія буде оброблена один раз. Вона зберігає стан обробки подій та може відновлювати обробку в разі відмови.
6. Легкість розробки та налагодження: Apache Samza надає високорівневий API для розробки обробників подій. Вона використовує мову програмування Java, що робить її доступною для широкого кола розробників. Крім того, вона надає інструменти для налагодження та моніторингу обробки подій.
7. Інтеграція: Samza добре інтегрується з іншими технологіями Apache, такими як Apache Kafka для передачі повідомлень, Apache Hadoop для зберігання та пакетної обробки даних, а також Apache YARN для управління ресурсами. Вона може працювати на кластерах, керованих Apache Mesos або Apache YARN.

Apache Samza використовується в різних сценаріях обробки даних в реальному часі, таких як аналітика в реальному часі, обробка журналів, виявлення шахрайства та системи моніторингу. Вона надає надійне та масштабоване рішення для розробки та використання програм обробки потоків з легкістю.

## Siddhi QL

*SiddhiQL є мовою запитів, що використовується у фреймворку WSO2 Siddhi для комплексної обробки подій у реальному часі.*

Siddhi QL (Siddhi Query Language) - це декларативна мова запитів, розроблена для роботи зі стрімінговими даними (streaming data). Вона широко використовується в системі комплексної обробки подій (Complex Event Processing, CEP) під назвою WSO2 Stream Processor.

Siddhi QL дозволяє визначати правила обробки стрімінгових даних шляхом опису умов, які повинні бути виконані, і дій, які повинні бути виконані при задоволенні цих умов. Вона має простий синтаксис, зрозумілий для розробників інтеграційних рішень та досвідчених користувачів.

Siddhi QL надає можливість виражати складні вимоги до стрімінгових даних, такі як умови фільтрації, агрегації, групування, вікна часу, з'єднання даних з різних джерел, маппінги, обчислення похідних подій тощо. Вона дозволяє швидко виявляти та реагувати на певні події, які виникають в стрімінгових даних, забезпечуючи реалізацію CEP-рішень.

### Розширення Siddhi QL

SiddhiQL, мова запитів для обробки подій, може бути розширена за допомогою розширень (extensions), які додають додаткові функціональність і можливості до базової мови. Розширення дозволяють розширити межі обробки подій і використовувати специфічні для домену функції та операції. Ось декілька розширень, які доступні для SiddhiQL:

1. Розширення математичних функцій: Деякі розширення додають математичні функції, такі як тригонометрія, логарифми, експоненти, обчислення векторів тощо. Ці розширення дозволяють виконувати складні обчислення над числовими даними в потоках подій.
2. Розширення обробки тексту: Деякі розширення надають функції для обробки текстових даних, такі як розбиття тексту на слова, пошук підрядків, виконання

операцій над рядками тощо. Ці розширення корисні для аналізу текстових даних у потоках подій.

3. Розширення роботи з часом: Деякі розширення додають функції для роботи з часовими даними, такі як отримання поточного часу, обчислення різниці між часовими мітками, перетворення форматів дати та часу тощо. Ці розширення полегшують обробку та аналіз подій, пов'язаних з часом.
4. Розширення машинного навчання: Деякі розширення дозволяють використовувати алгоритми машинного навчання, такі як класифікація, кластеризація, регресія тощо, для аналізу подій. Ці розширення дають можливість створювати складні моделі аналізу даних, які засновані на машинному навчанні.
5. Розширення інтеграції зі сторонніми системами: Деякі розширення додають підтримку інтеграції з іншими системами, такими як бази даних, системи сповіщень, веб-служби тощо. Ці розширення дозволяють отримувати та надсилати дані між SiddhiQL та іншими системами.

Ці розширення роблять SiddhiQL більш гнучкою та могутньою мовою для обробки подій, дозволяючи розробникам адаптувати її до конкретних вимог та сценаріїв використання.

**Виявлення аномалій за допомогою Siddhi QL**

SiddhiQL, мова запитів для обробки подій, може бути використана для виявлення аномалій в потоках подій. Для досягнення цієї мети можна використовувати різні підходи та конструкції мови SiddhiQL. Ось кілька способів виявлення аномалій за допомогою SiddhiQL:

1. Використання вбудованих агрегатних функцій: SiddhiQL має вбудовані агрегатні функції, такі як AVG, SUM, COUNT і т. д., які можна використовувати для обчислення статистичних метрик на основі потоку подій. Наприклад, ви можете визначити середнє значення деякого параметра і використовувати його для виявлення аномальних значень, які відхиляються від середнього.
2. Використання виразів віконного аналізу: SiddhiQL підтримує вирази віконного аналізу, які дозволяють обмежувати аналіз подій в певному вікні часу або кількості подій. Ви можете використовувати ці вирази для знаходження аномальних змін у поведінці або звернути увагу на події, які відбуваються занадто часто або занадто рідко.
3. Використання правил виявлення аномалій: Ви можете визначити свої власні правила виявлення аномалій, використовуючи конструкції SiddhiQL, такі як вказання шаблонів, умови фільтрації та дії. Наприклад, ви можете визначити правило, яке спрацьовує, якщо певний параметр перевищує певний поріг, і виконувати відповідні дії, які вказуєте.
4. Використання статистичних алгоритмів та машинного навчання: SiddhiQL може бути інтегрований з сторонніми бібліотеками для статистичного аналізу та машинного навчання, такими як Apache Spark або TensorFlow. Ви можете використовувати ці бібліотеки для розробки складніших моделей виявлення аномалій, які можуть аналізувати шаблони та залежності в потоках подій.

Виявлення аномалій за допомогою SiddhiQL вимагає детального розуміння вашого домену та відповідного налаштування запитів та правил. Додатковою перевагою SiddhiQL є його можливість підключення до різноманітних джерел даних та використання багатошарової обробки подій для складнішого аналізу.

## Event processing

Обробка подій (Event processing) - це процес збору, аналізу, інтерпретації та виконання дій на основі подій, які відбуваються в реальному часі або в стрімінгових даних. Події можуть бути спостереженнями або виникненнями конкретних подій, які мають важливість або цінність для певного контексту.

Обробка подій зазвичай включає в себе наступні кроки:

1. Збір подій: це включає отримання і реєстрацію подій з різних джерел, таких як сенсори, додатки, системи моніторингу тощо. Події можуть бути структурованими або некерованими.
2. Аналіз подій: після збору подій вони аналізуються для виявлення певних патернів, трендів або відхилень. Цей аналіз може включати застосування правил, моделей машинного навчання або статистичних алгоритмів для виявлення корисних знань або важливих подій.
3. Реагування на події: на основі аналізу подій виконуються певні дії або відправляються повідомлення для реагування на виникнення певних подій. Це може включати сповіщення, автоматичну обробку, запуск бізнес-процесів або взаємодію з іншими системами.

Обробка подій широко використовується в таких сферах, як фінанси, моніторинг і управління мережами, телекомунікації, логістика, маркетинг, медицина та багато інших. Вона дозволяє швидко реагувати на зміни, виявляти аномалії та визначати важливі події для покращення ефективності та прийняття рішень.

## CEP (Complex Event Processing)

Комплексна обробка подій (Complex Event Processing, CEP) - це технологія обробки стрімінгових даних, яка спеціалізується на виявленні та обробці складних подій, які виникають з великої кількості простих подій в реальному часі або в стрімінгових даних.

У традиційному сенсі події зазвичай розглядаються як окремі об'єкти, але CEP вирізняється своєю здатністю розуміти і обробляти більш складні шаблони та залежності між подіями. Вона дозволяє визначати складні умови, що включають послідовність, взаємозв'язок, часові вікна та агрегації, щоб виявити важливі зміни, тренди, аномалії або ситуації в реальному часі.

CEP може бути застосована в різних галузях, таких як фінанси, логістика, телекомунікації, безпека, моніторинг мереж, маркетинг і т. д. Вона дозволяє виявляти складні шаблони в подіях, наприклад, виявляти шахрайство на основі несподіваних послідовностей подій, визначати аномалії в операціях або моніторити трафік мережі для виявлення проблем.

CEP використовує різні алгоритми та техніки, такі як шаблони подій, складність подій, часові вікна, обробка високого обсягу даних (high volume data processing) та інші, для ефективної обробки стрімінгових даних та виявлення важливих подій. Деякі з популярних платформ, що надають підтримку CEP, включають WSO2 Stream Processor, Apache Flink, Esper, Apache Kafka Streams та інші.

### Приклади

Приклади використання CEP:

1. Фінансовий сектор: В банківській або фінансовій галузі CEP може використовуватись для виявлення шахрайства, спекуляції на фондовому ринку або незвичайних фінансових трансакцій. Він може аналізувати багато джерел даних, такі як транзакційні журнали, валютні курси, новини та соціальні медіа, для виявлення підозрілих або небезпечних активностей.
2. Інтернет в реальному часі: В онлайн-сервісах, таких як соціальні мережі, електронна комерція або веб-платформи, CEP може використовуватись для

виявлення популярних тем, трендів або поведінки користувачів. Він може аналізувати великі потоки даних, такі як повідомлення, лайки, коментарі, покупки тощо, для негайного реагування, персоналізації вмісту або підтримки рекомендацій.

3. Індустріальні системи: CEP може бути використаний для моніторингу та керування великими промисловими системами, такими як системи управління енергією, транспортні системи або машини на виробництві. Він може аналізувати дані з датчиків, прогнозувати аномальні стани, контролювати процеси та надавати автоматичні рекомендації або дії для оптимізації ефективності та безпеки.
4. Логістика та постачання: В галузі логістики та постачання CEP може використовуватись для моніторингу та управління ланцюжком постачання, виявлення затримок, прогнозування попиту або оптимізації маршрутів. Він може аналізувати дані про транспорт, вантажі, склади або попит, щоб забезпечити швидку та ефективну доставку товарів або послуг.
5. Виявлення аварій на машині: За допомогою CEP можна виявляти аварійні ситуації на машині, аналізуючи дані з датчиків, таких як температура, тиск, швидкість обертання тощо. Наприклад, якщо температура перевищує певне порогове значення протягом певного часу, система може спостерігати цю подію, спрацювати тривогу та виконати додаткові дії, наприклад, вимкнути машину.
6. Моніторинг транспортних систем: CEP може бути використаний для моніторингу транспортних систем, таких як автобуси або поїзди. Він може аналізувати дані про розклади, рух транспорту, затримки, пасажирський потік тощо. Наприклад, якщо автобус затримується більше певного порогового значення, система може спостерігати цю подію та повідомляти пасажирів про зміни розкладу або шукати альтернативні маршрути.

Це лише декілька прикладів використання CEP, і його потенціал застосування є дуже широким у багатьох галузях, де важлива реалізація аналітики в реальному часі та прийняття рішень на основі складних подій.

**Переваги**

Системи аналізу подій комплексного рівня (CEP) мають декілька переваг, які роблять їх цікавими інструментами для обробки та аналізу потоків подій у реальному часі. Ось деякі з них:

1. Реалізація реального часу: CEP дозволяє аналізувати потоки подій в реальному часі, надаючи миттєву відповідь на події та здатність реагувати негайно. Це дозволяє виявляти важливі шаблони, тренди або аномалії в потоках даних та вживати відповідних дій у найшвидший спосіб.
2. Складний аналіз подій: CEP надає потужні можливості для складного аналізу потоків подій, включаючи виявлення шаблонів, фільтрацію, агрегацію, кореляцію та інтеграцію даних з різних джерел. Це дозволяє здійснювати глибокий аналіз даних і отримувати цінні інсайти для прийняття рішень.
3. Реагування на події в реальному часі: CEP дозволяє реагувати на певні події або шаблони подій, спрацьовуючи визначені дії або сповіщення. Це дозволяє автоматизувати реакцію на певні події без затримки, що є особливо корисним для автоматизованих систем або ситуацій, де потрібна негайна реакція.
4. Інтеграція та розширення: CEP може бути легко інтегрований з існуючими системами та джерелами даних. Він також підтримує розширення та налаштування для виконання специфічних бізнес-правил та вимог. Це дозволяє використовувати CEP в різних сферах діяльності та налаштовувати його під конкретні потреби.
5. Візуалізація та моніторинг: CEP зазвичай надає можливості візуалізації та моніторингу потоків подій, що допомагає зрозуміти та аналізувати дані в зручний спосіб. Графіки, діаграми та панелі моніторингу дозволяють операторам та аналітикам відслідковувати та аналізувати стан системи та потоки даних.
6. Масштабованість: Деякі системи CEP мають вбудовану масштабованість, що дозволяє обробляти великі обсяги подій та підтримувати високу продуктивність. Це важливо для ситуацій, коли потоки подій швидко зростають або мають велику кількість одночасних подій.

Незважаючи на ці переваги, слід пам'ятати, що вибір системи CEP та її впровадження повинні враховувати специфіку бізнес-задач та потреби організації.

**Недоліки**

Хоча комплексний аналіз подій (Complex Event Processing, CEP) має багато переваг, він також має свої недоліки. Ось деякі з них:

1. Складність реалізації: Розробка та реалізація систем CEP може бути складною і вимагати глибокого розуміння домену, архітектури та дизайну системи. Встановлення та конфігурація системи CEP можуть також бути вимогливими завданнями.
2. Масштабованість: Деякі системи CEP можуть стикатися з обмеженнями масштабованості при великому обсязі подій або високій швидкості подачі подій. Обробка великого потоку подій та підтримка високої продуктивності можуть становити технічні виклики для систем CEP.
3. Складність управління правилами: В управлінні правилами для аналізу подій можуть виникати складнощі. Правила можуть бути складними, і керування набором правил та їх активування/деактивація може бути складним процесом. Крім того, зростання кількості правил може вплинути на продуктивність системи.
4. Обробка подій у реальному часі: Не всі системи CEP гарантують миттєву обробку подій у реальному часі. Деякі системи можуть мати деяку затримку у обробці та відповіді на події, що може не відповідати вимогам деяких використань.
5. Складність управління станом: Збереження та управління станом системи CEP може бути складним завданням, особливо в багатопотокових та розподілених середовищах. Відновлення та управління станом системи може становити технічні виклики.

Незважаючи на ці недоліки, системи CEP є потужними інструментами для аналізу подій у реальному часі та виявлення складних шаблонів та залежностей. Для успішного впровадження системи CEP необхідно ретельно розглянути вимоги,

архітектуру та вибір платформи, щоб забезпечити ефективну та надійну обробку подій.

**Альтернативні технології**

Існує кілька альтернативних технологій та платформ, які можуть служити альтернативою системам аналізу подій комплексного рівня (CEP). Ось декілька з них:

1. Системи потокової обробки (Stream Processing Systems): Ці системи спрямовані на обробку та аналіз потоків подій у реальному часі. Приклади таких систем включають Apache Flink, Apache Storm та Apache Spark Streaming. Вони надають можливості для розподіленої обробки потоків даних, агрегації, фільтрації та аналізу в реальному часі.
2. Бази даних потокових даних (Stream Data Platforms): Ці платформи дозволяють збирати, зберігати та аналізувати великі потоки подій у реальному часі. Приклади таких платформ включають Apache Kafka та Amazon Kinesis. Вони забезпечують масштабовану та надійну обробку потокових даних і можуть використовуватись для аналізу подій.
3. Комплексний аналіз даних (Complex Event Analytics): Ці системи спеціалізуються на аналізі складних подій, виявленні шаблонів та відносин між подіями. Вони надають аналітичні можливості для виявлення аномалій, прогнозування та іншого аналізу даних. Прикладами таких систем є Esper та Drools.
4. Системи обробки комплексних подій (Complex Event Processing Systems): Ці системи спеціалізуються на обробці та аналізі складних подій, але можуть мати інший підхід та функціональність порівняно з традиційними системами CEP. Прикладом такої системи є Oracle Event Processing.

Кожна з цих альтернативних технологій має свої переваги та особливості, і вибір залежить від конкретних потреб і вимог вашого проекту.

# Висновки

Розглянутий вище програмний комплекс (що включає Apache Kafka, Apache Hadoop YARN, Apache Samza і Siddhi QL) є повноцінним рішенням, що дозволяє:

1. Передавати події від джерела до точки опрацювання.
2. Опрацьовувати події з використанням високорівневої мови та із застосуванням сучасних концепцій (наприклад CEP).
3. Робота у режимі реального часу.

Одним із вагомих плюсів даної системи можна назвати те що майже кожен компонент можна замінити на альтернативу (майже кожен, тому що я не бачу реальних альтернатив для Apache Hadoop YARN). Так Apache Kafka можна замінити на Amazon Kinesis чи RabbitMQ, Samza на Spark чи Storm (теж від Apache), Siddhi QL на Esper, Drools, Spark Streaming (частина Spark) чи Flink CEP (частина Apache Flink). Відповідно це вимагатиме змін у коді, але не всього рішення, а тільки відповідного модуля. Вагомим аргументом для заміни певного компонента може стати його краща адаптованість до розв’язування конкретної поставленої задачі.

Іншими (але не менш значущими) перевагами є наступне:

1. Гнучке розподілення навантаження, масштабованість, і як наслідок можливість опрацювання великих об’ємів даних і підтримка роботи у real-time.
2. Стійкість до помилок, гарантія опрацювання кожної події – це може бути дуже важливим (якщо не основним аргументом) для певних видів використання (наприклад у фінансові чи медичній сфері).
3. Можливість використання комплексних підходів до опрацювання подій – CEP, часові вікна, об’єднання подій із різних топіків (JOIN в SiddhiQL).
4. Реалізація логіки опрацювання подій на високорівневій мові (SiddhiQL), можливість зміни коду без перекомпіляції/розгортання (методом заміни SiddhiQL на виконання усередині Samza App).
5. Загальна можливість до розширення системи, широкий потенціал існуючої екосистеми.
6. Кожна подія може бути оброблене декількома споживачами.

## Література


1. Neha Narkhede, Gwen Shapira, Todd Palino. (2017) Kafka: The Definitive Guide. O'Reilly Media, Inc.
2. Dylan Scott, Viktor Gamov, Dave Klein. (2022) Kafka in Action. Manning Publications.
3. Tom White. (2009). Hadoop: The Definitive Guide
4. Shumin Guo, Tanmay Deshpande. (2013) Hadoop Operations and Cluster Management Cookbook
5. Jay Kreps. (2014) I Heart Logs: Event Data, Stream Processing, and Data Integration.
6. Luckham, D. C. (2008). The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional.
7. Etzion, O., & Niblett, P. (2010). Event processing in action. Manning Publications.
8. Cardoso, J., & Sheth, A. P. (2005). Semantic e-business processes and rules. IEEE Internet Computing, 9(4), 16-23.
9. Zhang, Q., Zhang, L., & Chen, S. (2010). A survey of event processing systems. ACM Computing Surveys (CSUR), 43(3), 1-41.
10. Singh, M., Hsieh, W. C., & Lu, J. (2014). A survey of complex event processing. ACM Computing Surveys (CSUR), 46(3), 34.
11. Arasu, A., Babu, S., & Widom, J. (2006). The CQL continuous query language: semantic foundations and query execution. The VLDB Journal, 15(2), 121-142.
12. Carbone, P., Katsifodimos, A., Ewen, S., Markl, V., Haridi, S., & Tzoumas, K. (2015). Apache Flink: Stream and batch processing in a single engine. IEEE Data Engineering Bulletin, 38(4), 28-38.
13. Dindar, S., Fisher, M., & Yarvis, M. (2004). Event-Driven Computing: Applications, Architectures, and Algorithms. John Wiley & Sons.
14. Jin, J., Lin, C., Wu, H., & Yu, P. S. (2006). Event-driven QoS management for web services. IEEE Transactions on Knowledge and Data Engineering, 18(6), 814-828.
15. Luckham, D. C., Vera, D., Bryan, D., & Auguston, M. (2003). The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley.

16. Stonebraker, M., & Cetintemel, U. (2005). One size fits all: an idea whose time has come and gone. IEEE Data Engineering Bulletin, 28(1), 2-8.
17. Chandy, K. M., & Schulte, W. H. (2011). Event processing: Designing IT systems for agile companies. McGraw-Hill.
18. Cugola, G., & Margara, A. (2012). Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR), 44(3), 15.

# Додаток

Для створення Kafka топіка можна використовувати адміністративні інструменти Kafka, такі як **kafka-topics.sh** або **kafka-topics.bat**. Ось приклад використання цих інструментів для створення топіка.

Використання **kafka-topics.sh** або **kafka-topics.bat** з командного рядка:

*kafka-topics.sh --create --topic my_topic --bootstrap-server kafka:9092 --partitions 3 --replication-factor 2*

У цьому прикладі створюється топік з назвою "my_topic" з 3 партіціями та фактором реплікації 2. Перед виконанням потрібно замінити kafka:9092 на адресу та порт сервера Kafka.

## Приклад додатку на Go що створює і відправляє повідомлення у Kafka:

```go
package main

import (
 "encoding/json"
 "fmt"
 "log"

 "github.com/Shopify/sarama"
)

type (
 Message struct {
  ClientID     string `json:"clientID"`
  DepartmentID string `json:"departmentID"`
  AgentID      string `json:"agentID"`
  Message      string `json:"message"`
 }
)

func main() {
 // З'єднання з брокерами Kafka
 brokers := []string{"localhost:9092"}
 config := sarama.NewConfig()
 producer, err := sarama.NewSyncProducer(brokers, config)
 if err != nil {
  log.Fatal(err)
 }
 defer producer.Close()

 msg := &Message{
  ClientID:     "client_X",
  DepartmentID: "dep_A12",
  AgentID:      "123",
  Message:      "Hello, Kafka!",
 }

 msgBuf, err := json.Marshal(msg)
 if err != nil {
  log.Fatal(err)
 }

 // Надсилання повідомлень
 topic := "my_topic"
 // DepartmentID виористовується як ключ, по якому розраховується номер партіції, до якої
будуть надіслані повідомлення
 partitionKey := msg.DepartmentID
 message := string(msgBuf)

 kafkaMsg := &sarama.ProducerMessage{
  Topic: topic,
  Key:   sarama.StringEncoder(partitionKey),
  Value: sarama.StringEncoder(message),
 }

 // Відправка повідомлення в Kafka
 partition, offset, err := producer.SendMessage(kafkaMsg)
 if err != nil {
  log.Fatal(err)
 }

 fmt.Printf("Повідомлення надіслано до партіції %d з offset %d\n", partition, offset)
}
```

## Приклад Samza додатку для підрахунку слів - high-level API:

```
package com.kurs.samza.demo;

import org.apache.samza.application.StreamApplication;
import org.apache.samza.application.descriptors.StreamApplicationDescriptor;
import org.apache.samza.context.Context;
import org.apache.samza.operators.StreamGraph;
import org.apache.samza.operators.functions.MapFunction;

public class WordCountStreamApplication implements StreamApplication {
    @Override
    public void describe(StreamApplicationDescriptor appDescriptor) {
        // Визначення графу потоку
        StreamGraph streamGraph = appDescriptor.getStreamGraph();

        // Вихідний потік з Kafka
        streamGraph.getInputStream("inputStream")
                .map((MapFunction<String, String>) word -> word.toLowerCase())
                .flatMap((MapFunction<String, Iterable<String>>) line ->
Arrays.asList(line.split("\\W+")))
                .filter(word -> !word.isEmpty())
                .mapToTuple(word -> new Tuple2<>(word, 1))
                .groupBy(Tuple2::getF0, Tuple2::getF1, Serdes.String(), Serdes.Integer(),
"wordCount")
                .map((MapFunction<KV<String, Integer>, String>) kv -> kv.getKey() + ": " +
kv.getValue())
                .sendTo(streamGraph.getOutputStream("outputStream"));
    }

    @Override
    public void init(Context context) {
        // Ініціалізація
    }

    @Override
    public void close() {
        // Закриття
    }

    public static void main(String[] args) {
        WordCountStreamApplication app = new WordCountStreamApplication();
        app.run();
    }

}
```

## Приклад Samza додатку для підрахунку слів - low-level API:

```
package com.kurs.samza.demo;

import org.apache.samza.config.Config;
import org.apache.samza.system.IncomingMessageEnvelope;
import org.apache.samza.system.OutgoingMessageEnvelope;
import org.apache.samza.system.SystemStream;
import org.apache.samza.task.InitableTask;
import org.apache.samza.task.MessageCollector;
import org.apache.samza.task.StreamTask;
import org.apache.samza.task.TaskContext;
import org.apache.samza.task.TaskCoordinator;
import org.apache.samza.task.TaskCoordinator.RequestScope;

public class WordCountLowLevelStreamTask implements StreamTask, InitableTask {

    private SystemStream outputStream;

    @Override
    public void init(Config config, TaskContext context) {
        // Ініціалізація
        outputStream = new SystemStream("kafka", "outputStream");
    }

    @Override
    public void process(IncomingMessageEnvelope envelope, MessageCollector collector,
TaskCoordinator coordinator) {
        // Отримання даних з вхідного потоку
        String message = (String) envelope.getMessage();

        // Обробка даних
        String[] words = message.toLowerCase().split("\\W+");
        for (String word : words) {
            if (!word.isEmpty()) {
                // Публікація результату в вихідний потік
                collector.send(new OutgoingMessageEnvelope(outputStream, word, 1));
            }
        }

        // Повідомлення про успішне виконання завдання
        coordinator.commit(RequestScope.CURRENT_TASK);
    }

    public static void main(String[] args) {
        WordCountLowLevelStreamTask app = new WordCountLowLevelStreamTask ();
        app.run();
    }

}
```

## Приклад Samza додатку для підрахунку слів – з використанням Siddhi QL:

```
package com.kurs.samza.demo;

import org.apache.samza.application.StreamApplication;
import org.apache.samza.application.descriptors.StreamApplicationDescriptor;
import org.apache.samza.context.Context;
import org.apache.samza.operators.StreamGraph;
import org.apache.samza.operators.functions.MapFunction;
import org.apache.samza.system.IncomingMessageEnvelope;
import org.apache.samza.system.OutgoingMessageEnvelope;
import org.apache.samza.system.SystemStream;
import org.apache.samza.system.SystemStreamPartition;
import org.apache.samza.task.InitableTask;
import org.apache.samza.task.MessageCollector;
import org.apache.samza.task.StreamTask;
import org.apache.samza.task.TaskContext;
import org.apache.samza.task.TaskCoordinator;
import org.apache.samza.task.TaskCoordinator.RequestScope;

import org.wso2.siddhi.core.SiddhiManager;
import org.wso2.siddhi.core.event.Event;
import org.wso2.siddhi.core.stream.input.InputHandler;

public class WordCountSiddhiQLStreamTask implements StreamTask, InitableTask {

    private SiddhiManager siddhiManager;
    private InputHandler inputHandler;

    @Override
    public void init(Config config, TaskContext context) {
        // Ініціалізація Siddhi
        siddhiManager = new SiddhiManager();
        String siddhiApp = "define stream inputStream (word string, count int); " +
                "from inputStream#window.timeBatch(1 sec) " +
                "select word, sum(count) as totalCount " +
                "insert into outputStream;";
        siddhiManager.createSiddhiAppRuntime(siddhiApp);

        // Ініціалізація Apache Samza
        SystemStreamPartition inputStreamPartition = new SystemStreamPartition("kafka",
"inputStream", 0);
        inputHandler = siddhiManager.getInputHandler("inputStream");

        // Підписка на вхідні дані
        context.registerOffsetTask(inputStreamPartition.getSystemStream(), () -> {
            // Повертаємо оффсет, якщо потрібно керувати оффсетами для перезавантаження
        });
    }

    @Override
    public void process(IncomingMessageEnvelope envelope, MessageCollector collector,
TaskCoordinator coordinator) {
        // Отримання даних з вхідного потоку
        String message = (String) envelope.getMessage();

        // Надсилання даних в Siddhi CEP для обробки
        try {
            inputHandler.send(new Object[]{message, 1});
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Повідомлення про успішне виконання завдання
        coordinator.commit(RequestScope.CURRENT_TASK);
    }

    @Override
    public void window(MessageCollector collector, TaskCoordinator coordinator) {
```

```
        // Обробка віконних результатів в Siddhi CEP
        Event[] events = siddhiManager.query("from outputStream select *;");
        for (Event event : events) {
            String word = (String) event.getData()[0];
            long totalCount = (Long) event.getData()[1];

            // Відправка результату до вихідного потоку Apache Samza
            collector.send(new OutgoingMessageEnvelope(new SystemStream("kafka",
"outputStream"), word + ": " + totalCount));
        }

        // Повідомлення про успішне виконання завдання
        coordinator.commit(RequestScope.CURRENT_TASK);
    }

    @Override
    public void close() {
        // Закриття Siddhi CEP
        siddhiManager.shutdown();
    }

    public static void main(String[] args) {
        WordCountSiddhiQLStreamTask app = new WordCountSiddhiQLStreamTask();
        app.run();
    }

}
```