Лекція 16.
Розподілена потокова платформа Kafka.
Переваги Cassandra

План лекції

16.1. Проблема прийому даних.
16.2. Розподілена потокова платформа Kafka.
16.3. Переваги Cassandra.

16.1. Проблема прийому даних

Веб-сервіси, що допомагають надійно і з заданими інтервалами обробляти дані і переміщати їх між різними обчислювальними сервісами, називаються pipeline та виконують приймання, зберігання та оброблення даних (рис.16.1). Для кожного з цих компонентів існує багато програмних платформ, які використовуються для виконання кожного завдання. Залежно від типу даних, типу прийому та вимог до обчислень, компоненти кожного pipeline даних унікальні, часто складаються разом і коригуються для роботи один з одним.

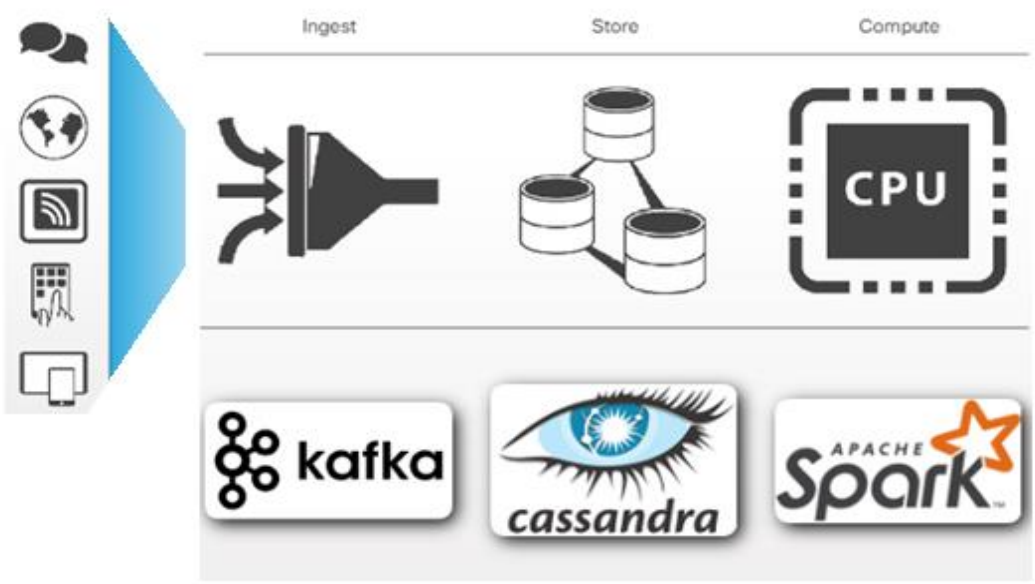

Рис. 16.1. Pipeline великих даних [1]

Великі дані часто надходять із багатьох різних джерел. Значна частина цих даних є потоковою і повинна прийматися в режимі реального часу. Масштабованість також є проблемою. Коли більше пристроїв підключається, більше даних потрібно приймати. Якість цих даних також може

бути проблемою. Дані можуть бути структурованими, неструктурованими або мати дуже складні формати. Існує багато різних інструментів для збору та переміщення великої кількості даних з різних джерел до сховища даних.

16.2. Розподілена потокова платформа Kafka

Для передачі даних у режимі реального часу необхідно використовувати розподілену потокову платформу, таку як Kafka. Характеристики Kafka:

- Потоки записів публікуються та підписуються на (pub-sub) аналогічно, як у системі обміну повідомленнями підприємства.
- Потоки записів стійкі до відмов через розподілене зберігання.
- Потоки записів обробляються, як вони трапляються, в режимі реального часу.

Kafka використовується для передачі потокової передачі даних у режимі реального часу між різними системами та програмами. Kafka також використовується для перетворення та реагування на потоки даних через додатки в режимі реального часу. Kafka була розроблена LinkedIn, але стала програмним забезпеченням з відкритим кодом у 2011 році. Cisco Systems, Netflix та eBay – це лише декілька підприємств, які використовують Kafka.

Розглянемо основні положення Kafka.

- Kafka працює на одному або декількох серверах як кластер. Сервери в кластері відомі як брокери.
- Кластер зберігає потоки записів у групах, що називаються темами.
- Кожен запис містить ключ, значення та часову позначку.

У Kafka є чотири основних API (рис. 16.2):

- Виробники. API, де потік записів публікується додатком на теми Kafka.
- Потокові процесори. API, де споживаються потоки входу з тем і можуть вироблятися потоки виводу до тем.
- З'єднувачі. API, де теми Kafka підключаються до існуючих систем та програмних додатків.
- Споживачі. API, де створюється потік записів у темах.

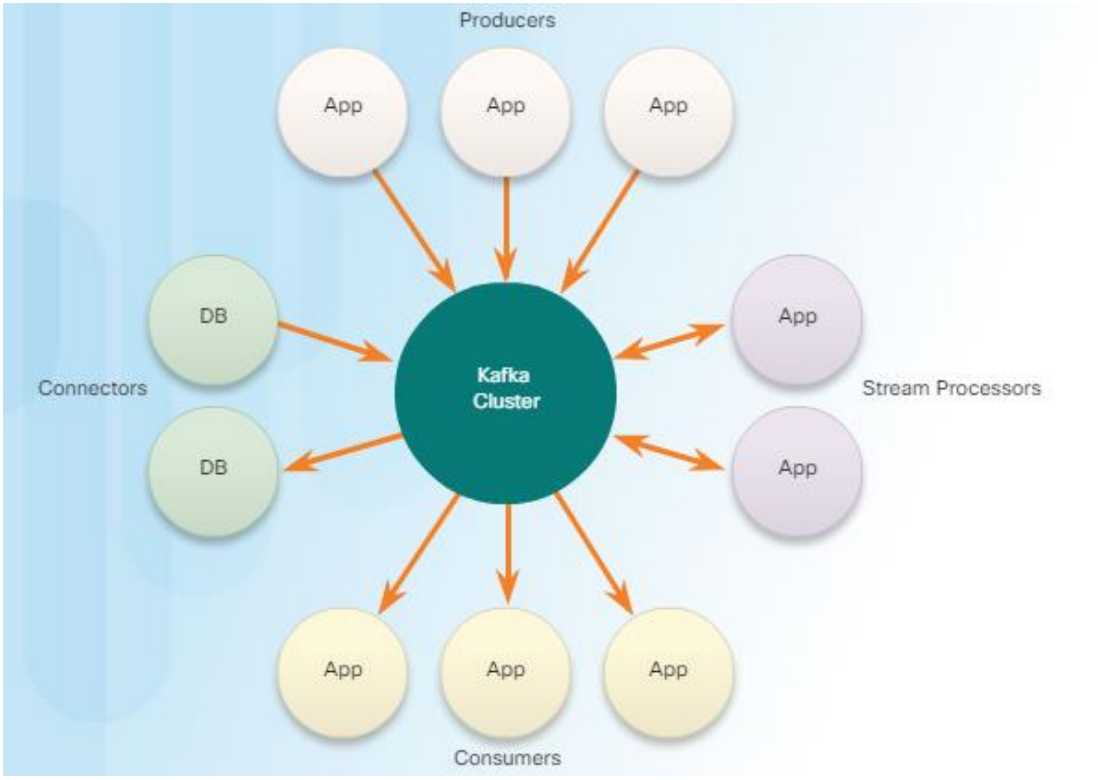

Рис. 16.2. Компоненти API Kafka [1]

Kafka діє як система обміну повідомленнями для централізації зв'язку між усіма виробниками та споживачами даних у системі. Kafka використовує розподілену обробку величезної кількості даних та масштабування, забезпечує відмовостійкість обладнання та програмного забезпечення. Передача повідомлень в Kafka відбувається двома способами (рис.16.3):

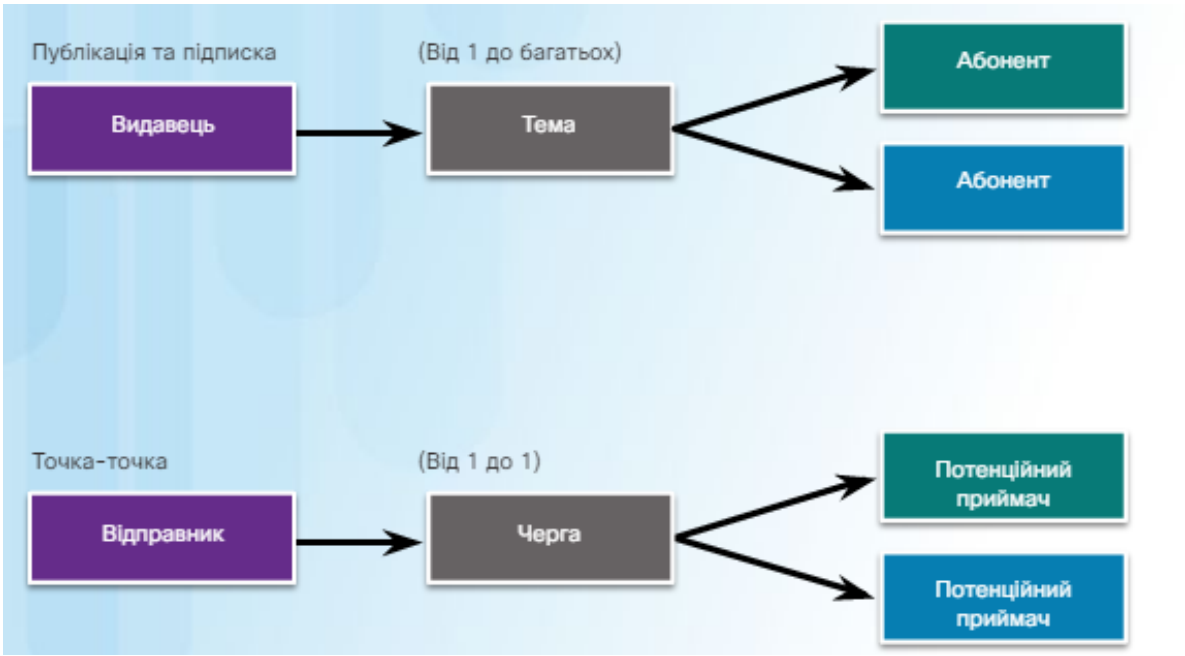

Рис. 16.3. Методи передачі повідомлень [1]

- **Опублікувати та підписатися** – запитувані повідомлення передаються всім споживачам.
- **Точка-в-точку** – кілька споживачів читають повідомлення з сервера. Кожне з цих повідомлень надходить до одного із споживачів.

Kafka – це набагато більше, ніж просто сервер обміну повідомленнями. Kafka працює аналогічно розподіленій базі даних.

Повідомлення, написані Kafka, копіюються на багато серверів і записуються на диск. Завдяки розподіленому проектуванню Kafka має високу доступність, підтримує автоматичне відновлення даних та відрізняється високою стійкістю до відмов мережі.

Kafka відрізняється від традиційних брокерів повідомлень використанням журналів транзакцій. Кожна тема складається з набору журналів, які називаються **розділами**. Виробники додають ці журнали, і споживачі можуть читати журнали, коли потрібно. **Теми** – це дані, які реплікуються багатьма брокерами для досягнення відмовостійкості.

Kafka може управляти високою пропускною здатністю завдяки тому, що багато задач покладаються на виробників та споживачів. Це зберігає брокери у легкій вазі та робить Kafka бажаним інструментом для багатьох платформ IoT. Завдяки величезній кількості даних, що надходять від датчиків та інших пристроїв у режимі реального часу, Kafka може регулювати прийом даних та надавати дані для кількох споживачів одночасно.

Згідно оцінок великих даних IBM, "кожен день ми створюємо 2,5 квинтільйонних байта даних" кожного дня, оскільки щохвилини:

- завантажується понад 300 годин відео YouTube;
- надсилається понад 3,5 мільйона текстових повідомлень;
- передається понад 86 тисяч годин відео Netflix;
- оцінюється понад 4 мільйони публікацій у Facebook.

Завдяки цьому постійно створюється величезний обсяг даних, навіть із хмарними сховищами, які доступні таким компаніям, як Amazon, Google, Microsoft, безпека даних стає великою проблемою.

Рішення Big Data повинні бути захищеними, мати високу стійкість до відмов, вміти горизонтально масштабувати та використовувати реплікацію, щоб дані не втрачалися.

Big Data – це термін для величезного обсягу даних, який ми постійно створюємо з величезної кількості джерел даних. Ці дані повинні бути релевантними, чистими та безпечними. Існує щонайменше п’ять проблем із зберіганням даних із Big Data:

- **Управління** – існує мало стандартів обміну даними та тисячі інструментів управління даними.
- **Безпека** – автентифікацію, доступ та облік важко забезпечити.
- **Неструктуровані дані** – неструктуровані дані важко аналізувати та шукати.

16.3. Переваги Cassandra

Cassandra – це система управління розподіленими базами даних з відкритим кодом NoSQL. База даних NoSQL не містить схем і не використовує традиційні методи зберігання та отримання даних, таких як реляційні бази даних, має просту конструкцію, підтримує горизонтальне масштабування та забезпечує більший контроль доступності. Cassandra може бути повністю розповсюджена та розгорнена по всьому світу, якщо потрібно та надає децентралізовану базу даних без жодної точки відмови.

Система управління розподіленими базами даних Cassandra розроблена компанією Facebook у 2008 році. Одна з речей, яка робить Cassandra швидкою, це те, що вона використовує послідовне читання та запис. Це дуже зручно для роботи з даними часових рядів в рішеннях IoT. Замість додавання або видалення даних у файлі створюється новий файл і видаляються старі файли.

Основні характеристики Cassandra:

- **Поширення даних** – дані можна копіювати в декількох центрах обробки даних.

- **Підтримка транзакцій** – підтримує атомність, консистенцію, ізоляцію та довговічність (atomicity, consistency, isolation, durability, ACID).
- **Еластична масштабованість** – коли потрібно більше даних або додається більше клієнтів, може бути додано більше обладнання для масштабування за потребою.
- **Швидке лінійне масштабування** – коли кількість вузлів збільшується в кластері, пропускна здатність збільшується, підтримуючи швидку реакцію.
- **Швидкий запис** – Cassandra виконує швидкий запис, зберігаючи сотні терабайт даних.
- **Завжди ввімкнено** – Cassandra завжди доступна навіть при збоях у апаратному та / або мережевому режимі.
- **Гнучкість зберігання даних** – підтримуються неструктуровані, структуровані та напівструктуровані дані.

Apache Hadoop визначає HDFS як "первинну систему зберігання, яка використовується програмами Hadoop", що дозволяє виконувати "надійні та швидкі обчислення".

Первинний NameNode в кластері регулює файлову систему та доступ до даних. У кластері також є DataNodes, часто одна фізична машина для обробки доданого сховища.

Дані зберігаються у файлах, поділяються на блоки та поширюються по DataNodes. HDFS копіює ці блоки на два додаткові сервери за замовчуванням.

Cassandra використовує файлову систему Cassandra File System (CFS).

Кожен вузол кластера має однакову однорангову реалізацію.

Кластери зберігають дані в реальному часі і аналітичні операції можуть бути виконані на цих даних (рис.16.4).

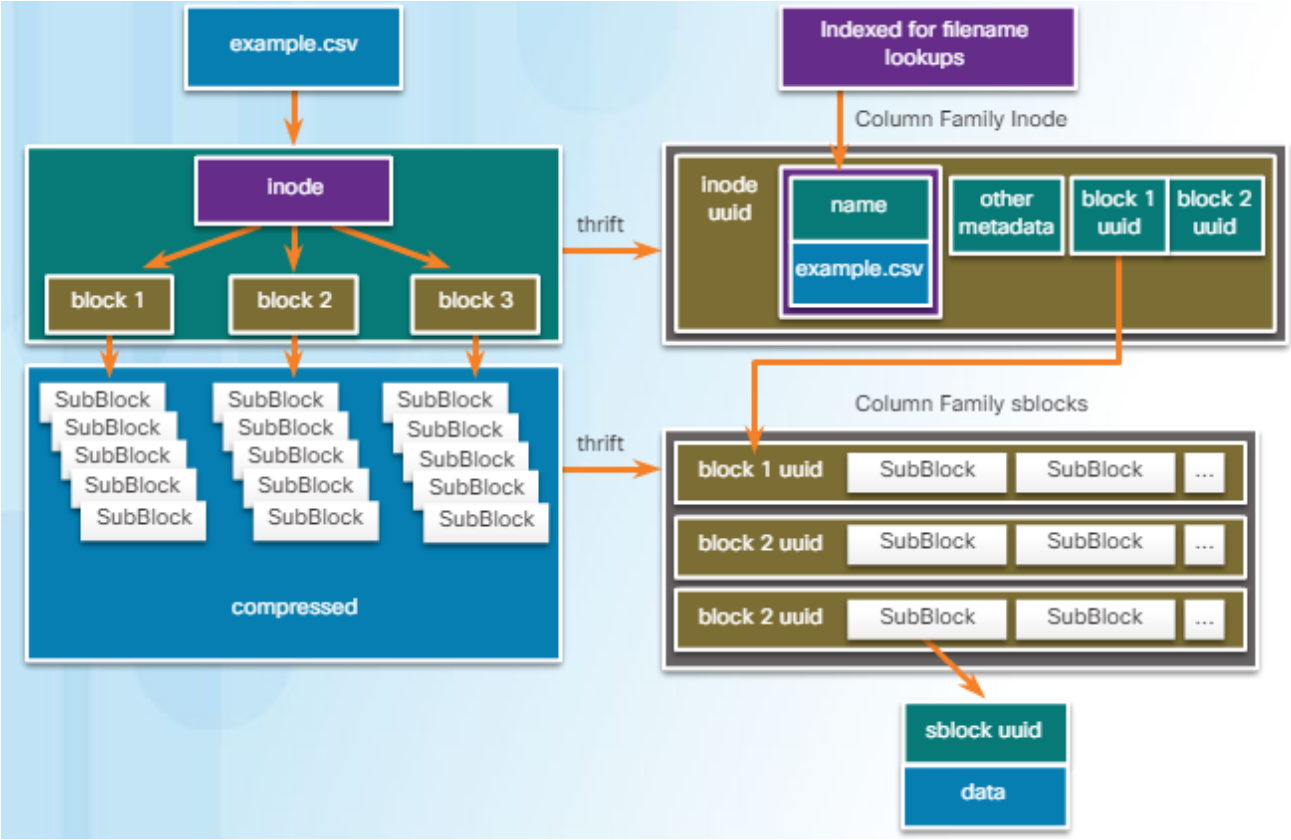

Рис. 16.4. Cassandra File System [1]

Вбудована реплікація копіює дані серед усіх реальних аналітичних та пошукових вузлів.

Є дві сімейства стовпців, схожі на таблиці RDBMS, які містять дані.

Дані в цих сімействах стовпців реплікуються через кластер для забезпечення захисту даних. Сімейства стовпців схожі з двома основними службами HDFS.

Сімейство стовпців **inode** займає місце служби HDFS NameNode.

Сімейство стовпців **sblocks** займає місце служби HDFS DataNode. Ці стовпці зберігають вміст будь-якого файлу.

Переваги використання CFS у порівнянні з HDFS:

- **Краща доступність** – рішення спільного зберігання не потрібно. Чим більше вузлів і кластерів, тим краща доступність. Це також можна покращити, збільшивши коефіцієнт реплікації, який визначає, скільки вузлів отримають репліковані дані.
- **Апаратна підтримка** – не потрібні спеціальні сервери та спеціальні мережеві пристрої для CFS.

- **Автоматичне відновлення** – вузли, кластери, навіть центри обробки даних можуть вийти з ладу, а дані про вузли та кластери залишатимуться доступними в інших місцях.
- **Інтеграція даних** – усі дані, записані в Cassandra, реплікуються як в аналітичні, так і в пошукові вузли. Це дозволяє одночасно виконувати аналітичні, пошукові роботи та завдання в режимі реального часу, не впливаючи один на одного.
- **Легше розгортання** – кластери легко налаштувати і можуть працювати лише за кілька хвилин.
- **Підтримується декілька центрів обробки даних** – CFS може запускати одну базу даних у кількох центрах обробки даних. Доступні інструменти, що дозволяють кожному центру обробки даних мати локальну копію всіх даних у базі даних. Аналітичні завдання можуть виконуватися в декількох центрах обробки даних одночасно.

HDFS – це відмінний вибір, коли для програми Hadoop потрібне недороге рішення для зберігання даних з акцентом на зберігання даних.

CFS може виконувати аналітику даних, що надходять із великих програм, що інтегруються в бази даних та СУБД.

Apache Cassandra повністю реалізує принципи доступності (Availability) і стійкості до поділу (Partition tolerance), забезпечуючи коректний відгук на будь-який запит і простоту масштабування. Однак узгодженість (Consistency), тобто несуперечливість даних вважається слабким місцем цієї розподіленої СУБД в зв'язку з її децентралізацією, коли на багатьох вузлах зберігаються різні репліки однієї і тієї ж інформації. Для усунення цієї проблеми використовується механізм налаштування рівнів узгодженості.

На відміну від NoSQL-бази даних для Big Data, Apache HBase, усі вузли кластера Cassandra рівноцінні – клієнти можуть з'єднуватися з будь-яким з них для запису і читання. Виконання запиту починається з його координації , щоб за допомогою ключа і міток визначити, на яких вузлах кластера знаходяться потрібні дані і відправити запит саме туди. Вузол, що виконує координацію,

називається **координатором (coordinator)** , а вузли, які обрані для збереження запису з даними ключем – **вузлами-реплік (replica nodes)**. Фізично координатором може бути один з вузлів-реплік.

Доступність даних безпосередньо залежить від рівня узгодженості операцій читання і запису, так як він визначає, скільки вузлів-реплік може відмовити при підтвердженні успішного виконання цих операцій. Наприклад, якщо рівень реплікації менший, ніж сума вузлів, з яких прийшло підтвердження про успішний записи даних, і з яких відбувається читання, тобто **гарантія суворої узгодженості (strong consistency)** . Сувора узгодженість гарантує, що після запису нового значення завжди буде прочитано. Інакше можлива ситуація, коли в результаті читання СУБД поверне застарілі дані. Для боротьби з цим в Apache Cassandra є **механізм підсумкової узгодженості (eventual consistency)** , який поширює дані по вузлах-реплік після того, як закінчиться координаційне очікування. Якщо при цьому будуть доступні не всі вузли-репліки, то доведеться задіяти інші засоби відновлення даних, зокрема, **читання з виправленням** і запускається вручну **анти-ентропійне відновлення вузла (anti-entropy node repair)**.

Рис. 16.5. Розподілений запис даних в кластері Apache Cassandra [2]

При запису даних рівень узгодженості визначає кількість вузлів-реплік, з яких буде очікуватися підтвердження вдалого закінчення операції – сигналу того, що дані успішно записані. Для запису існують такі рівні узгодженості:

- **ANY (0)** – дає можливість записати дані, навіть якщо всі вузли-репліки не відповідають. Координатор отримує відповідь від будь-якого одного вузла-реплік або дані збережуться за допомогою механізму спрямованої відправки (hinted handoff) на координатора.
- **ONE (1)** – координатор шле запити всіх вузлів-реплік, але повертає управління користувачеві, дочекавшись підтвердження від будь-якого першого вузла;
- **TWO (2)** – координатор чекає підтвердження від двох перших вузлів, перш ніж повернути управління;
- **THREE (3)** – координатор чекає підтвердження від трьох перших вузлів, перш ніж повернути управління;
- **QUORUM (4)** – координатор чекає підтвердження записів від більш ніж половини вузлів-реплік, а саме *round (N/2)+1*, де *N* – рівень реплікації;
- **ALL (5)** – координатор чекає підтвердження від всіх вузлів-реплік;
- **LOCAL_ QUORUM (6)** – координатор чекає підтвердження від більш ніж половини вузлів-реплік в тому ж центрі обробки даних, де розташований координатор. Це дозволяє позбутися затримок, пов'язаних з пересиланням даних у інші датацентри.
- **EACH_ QUORUM (7)** – координатор чекає підтвердження від більш ніж половини вузлів-реплік в кожному центрі обробки даних.

У разі читання рівень узгодженості визначає кількість вузлів-реплік, з яких буде зчитана інформація:

- **ONE (1)** – координатор шле запити до найближчого вузла-репліки, читаючи інші з метою виправлення (read repair) із заздалегідь заданою в конфігурації ймовірністю;
- **TWO (2)** – координатор шле запити до двох найближчих вузлів, вибираючи значення з більшою часовою позначкою;

- **THREE (3)** – координатор шле запити до трьох найближчих вузлів, вибираючи значення з більшою часовою позначкою;
- **QUORUM (4)** – збирається кворум, тобто координатор шле запити до більш ніж половині вузлів-реплік, а саме *round (N/2)+1* , де *N* – рівень реплікації;
- **ALL (5)** – координатор повертає дані після прочитання з усіх вузлів-реплік;
- **LOCAL_QUORUM** (6) – збирається кворум вузлів в тому датацентрі, де відбувається координація, і повертаються дані з останньою міткою часу;
- **EACH_QUORUM** (7) – координатор повертає дані після зборів кворуму в кожному з датацентрів.

Підводячи підсумок можливим рівням узгодженості в Apache Cassandra, можна зробити наступні висновки:

- **QUORUM** на читання і на запис забезпечить сувору узгодженість і баланс між затримкою на виконання цих операцій;
- strong consistency також буде досягнута під час запису ALL, і читанні ONE. Однак, в цьому випадку дані зчитуються швидше і з більшою доступністю. Для запису будуть потрібні всі робочі вузли-реплік.
- Зворотний випадок (**запис ONE, читання ALL**) забезпечує сувору узгодженість, запис буде виконуватися швидше і з більшою доступністю, тому що буде потрібно підтвердження про успішне закінчення операції лише з одного вузла.
- **При відсутності вимог про сувору узгодженість** можна прискорити операції читання і запису, а також покращити доступність за рахунок виставлення менших рівнів узгодженості.

Висновок до лекції 16

Розподілена потокова платформа Kafka відрізняється від традиційних посередників повідомлень використанням журналів транзакцій для передачі потокових даних у режимі реального часу між різними системами та програмами.

Cassandra – це система розподілених баз даних з відкритим кодом NoSQL. Cassandra використовує файлову систему Cassandra File System (CFS). Кожен вузол кластера має однакову однорангову реалізацію. Кластери зберігають дані в реальному часі і аналітичні операції виконуються на цих даних.

Питання для закріплення

1. Які проблеми прийому даних ви знаєте?
2. Для чого використовується Kafka?
3. Яке призначення та переваги Cassandra?

Список рекомендованої літератури

IoT Fundamentals: Big Data & Analytics // Електронний ресурс. Режим доступу: https://www.netacad.com/courses/iot/big-data-analytics

Рівні узгодженості Apache Cassandra для розподіленої обробки Big Data // Електронний ресурс. Режим доступу: https://www.bigdataschool.ru/blog/cassandra-consistency-levels.html

What is Cassandra? // Електронний ресурс. Режим доступу: https://cassandra.apache.org/

About the Cassandra File System (CFS) – deprecated // Електронний ресурс. Режим доступу: https://docs.datastax.com/en/dse/5.1/dse-dev/datastax_enterprise/ analytics/ cfsAbout.html

Лекція 17.
Платформа Apache Spark

План лекції

17.1. Проблема обчислювальної функції.
17.2. Технологія Spark.
17.3. Порівняння Spark та MapReduce.
17.4. Spark і sparklyr для роботи з великими даними в R.

17.1. Проблема обчислювальної функції

Важливим завданням, з яким стикаються обчислення Big Data, є розмір наборів даних, що використовуються у різних галузях. Наприклад, генетичне секвенування (цифрове представлення геному людини) в останні роки стало