УДК 004.056.5

Д.Д. Левченко

Національний аерокосмічний університет імені М.Є. Жуковського «ХАІ», Харків

АНАЛІЗ ФАКТОРІВ, ЩО ВПЛИВАЮТЬ НА ПРОДУКТИВНІСТЬ КЛАСТЕРУ НЕРЕЛЯЦІЙНОЇ БАЗИ ДАНИХ CASSANDRA

Головною ідеєю статті є аналіз факторів, що впливають на продуктивність кластеру Cassandra. У роботі була розглянута архітектура кластеру нереляційної бази даних, проаналізовані можливості такої архітектури. У статті перелічені основні фактори, які впливають на продуктивність кластеру. Особливу увагу було приділено таким факторам як кешування та довговічність.

Ключові слова: *база даних, Cassandra, архітектура, кластер, продуктивність, commitlog, операції читання, операції запису, кеш, кешування, довговічність, fsync.*

Вступ

Актуальність. Аналіз літератури. До появи реляційної моделі розробники спробували інші моделі, такі як ієрархічні моделі і спрямований граф. Заснована на SQL реляційна модель - яка сьогодні є стандартом де-факто - залишається переважаючою близько 30 років. Це чудове досягнення, враховуючи коротку історію і швидкі темпи розвитку обчислювальної техніки. Реляційна модель настільки добре вкоренилася, що протягом багатьох років вибір способу зберігання даних для додатків був очевидним.

Однак такі явища, як зростання бази користувачів систем, мобільні пристрої, розширене присутність користувачів в Інтернеті, хмарні обчислення і багатоядерні системи, привели до появи все більш великомасштабних систем. Хайтек-компанії, такі як Google і Amazon, одними з перших зіткнулися з цими проблемами масштабу. Незабаром вони виявили, що реляційні бази даних не оптимальні для підтримки великомасштабних систем [3].

У міру зростання обсягу банку даних сильно знижується швидкодія системи. Одним із шляхів зменшення часу доступу до даних є розміщення бази даних в оперативній пам'яті. Ця техніка дозволяє отримати виграш у швидкодії до 100 разів.

Для вирішення цієї проблеми (big data) розроблена спеціальна різновид баз даних NoSQL.

Технологія NoSQL (наприклад, Cassandra) не призначена замінити реляційні бази даних, скоріше вона допомагає вирішити проблеми, коли обсяг даних стає занадто великий. NoSQL часто використовує кластери недорогих стандартних серверів. Це рішення дозволяє знизити вартість на гігабайт в секунду в кілька разів [11].

Щоб вирішити цю проблему, Google і Amazon запропонували два альтернативних рішення: Big Table і Dynamo, в яких вони відмовилися від гарантій, що надаються реляційною моделлю даних, заради більш високої масштабованості. Пізніше ці відкриття формалізувала «теорема CAP» Еріка Брюєра. У ньому записано, що для масштабованих систем узгодженість даних, надійність і стійкість до поділу - взаємовиключні властивості, і неможливо побудувати систему, що володіє всіма цими властивостями. Незабаром, спираючись на ранні роботи Google і Amazon, а також накопичені знання в області масштабованих систем, розробники запропонували новий клас систем зберігання даних. Їх назвали системами NoSQL. Спочатку це означало "do not use SQL if you want to scale" (не використовуйте SQL там, де потрібно масштабування), а пізніше розшифровку замінили на "not only SQL" (не тільки SQL), підкреслюючи, що крім SQL-рішень існують і інші [5].

Відомо безліч систем NoSQL, і кожна виключає або змінює ті чи інші аспекти реляційної моделі. Варто зазначити, що жодне з рішень NoSQL не працює у всіх сценаріях. Кожне перевершує реляційні моделі і масштабується для деякої обмеженої області застосування.

Cassandra широко використовується в усьому світі, і її використання зростає з весь час. Такі компанії як Netflix, eBay, Twitter, Reddit, а також Ooyala використовують Cassandra як частину своєї архітектури, і це має вирішальне значення день у день роботи цих організацій.

На сьогоднішній день, невеликий відкритий кластер Cassandra за кількістю машин має 300 ТБ даних на охоплюють 400 машин [6].

Через те що Cassandra має високошвидкісну обробку даних, вона добре працює для безлічі додатків.

Це означає, що вона підходить для обробки проектів у високошвидкісному світі рекламних технологій в режимі реального часу в великих обсягах світу аналітики великих даних і все між ними. [11]

Метою даної статті є аналіз факторів, що впливають на продуктивність кластеру нереляційної бази даних Cassandra.

1. Архітектура кластеру Cassandra

Архітектура кластера Cassandra показана на рис. 1. Відразу видно, що Cassandra - розподілена система. Вона складається з декількох вузлів і розподіляє дані між цими вузлами (або секціонуючою їх по термінології баз даних).

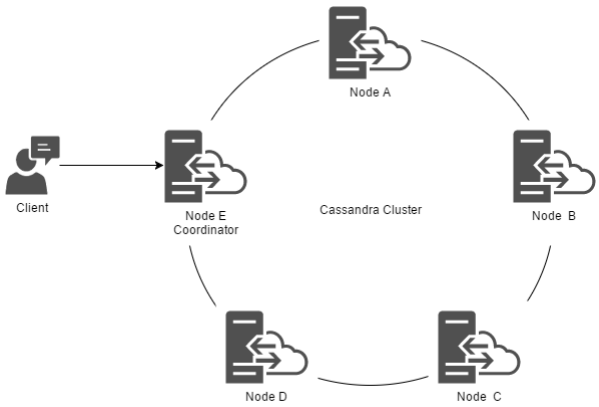

Рис. 1. Архітектура кластеру Cassandra

Для розподілу елементів даних по вузлах Cassandra використовує послідовне хешування. Простіше кажучи, Cassandra використовує хеш-алгоритм для обчислення хеш-значень ключів кожного елемента даних, що зберігається в Cassandra (ім'я стовпця, ID рядки і т.п.). Діапазон хеш-значень або всі можливі хеш-значення (т.зв. простір ключів) розподіляється між вузлами кластера Cassandra. Потім Cassandra призначає кожному елементу даних свій вузол, і цей вузол відповідає за зберігання і управління цим елементом даних. Докладний опис архітектури Cassandra (рис. 2) міститься в документі "Cassandra - A Decentralized Structured Storage System"[5].

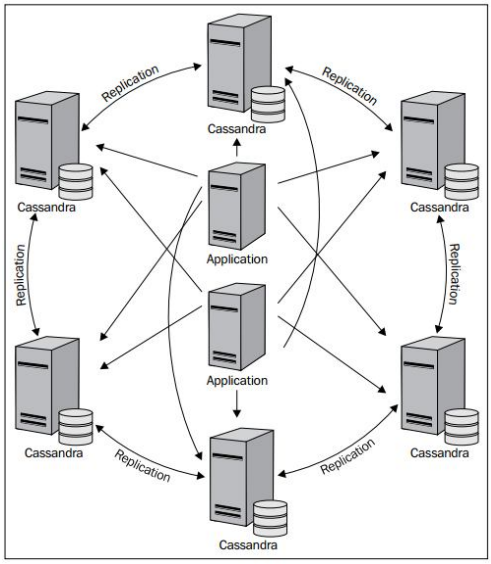

Рис. 2. Архітектура Cassandra

Така архітектура надає наступні можливості:

– Cassandra розподіляє дані між вузлами прозоро для користувачів. Будь-який вузол може приймати будь-який запит (читання, запис або видалення), і якщо дані зберігаються не в цьому вузлі, перенаправляє його в потрібний вузол;

– користувачі можуть визначити необхідну кількість реплік, і Cassandra прозоро забезпечить створення реплік і управління ними;

– настроюється узгодженість: при зберіганні і зчитуванні даних користувачі можуть вибирати рівень узгодженості по кожній операції. Наприклад, якщо під час запису або читанні використовується рівень узгодженості «кворум», то дані записуються і зчитуються більш ніж з половини вузлів кластера. Підтримка настроюється узгодженості дозволяє вибрати рівень узгодженості, найбільш підходящий для даного випадку;

– Cassandra забезпечує дуже швидкий запис, швидшу, ніж читання, зі швидкістю передачі даних порядку 80-360 МБ / с на вузол. Це досягається за допомогою двох підходів:

– Cassandra зберігає більшу частину даних в оперативній пам'яті відповідального вузла, і будь-які оновлення виконуються в пам'яті, а потім записуються в постійну систему зберігання (файлову систему) ледачим методом. Проте, щоб уникнути втрати даних Cassandra реєструє всі транзакції в журналі фіксації транзакцій на диску. На відміну від поновлення елементів даних на диску, записи в журнали фіксації можуть тільки додаватися, що виключає затримку обертання диска;

– якщо не потрібна повна узгодженість записів, Cassandra записує дані в достатню кількість вузлів без дозволу конфліктів невідповідності, які дозволяються тільки при першому зчитуванні. Цей процес називається «ремонтом при читанні».

Результуюча архітектура добре масштабується. Можна побудувати кластер Cassandra з десятками або сотнями вузлів, здатний обробляти терабайти або петабайт даних. [10, 11]

2. Продуктивність кластеру Cassandra

Data Storage. Є два набори файлів, в які Cassandra записує в якості частини володіння операціями оновлення: журнал фіксації (commit log) і файл даних. Їх різні цілі необхідно враховувати для того, щоб зрозуміти, як поводитися з ними під час конфігурації.

Commit log можна розглядати як короткострокове зберігання. Як тільки Cassandra отримує оновлення, кожне значення запису записуються безпосередньо в журнал фіксацій у вигляді рядка, послідовно дописуючи файл. Якщо припинити роботу бази даних або вона виходить з ладу несподівано, commit log може гарантувати, що дані не втрачаються. Це тому, що в наступний раз, коли запускається вузол, то журнал фіксації відтворюється. Насправді, це

єдиний раз, коли commit log читається; клієнти ніколи не читають з нього. Але нормальна операція запису журналу фіксації блоків може зашкодити продуктивності, як вимагають клієнти, які чекають завершення запису.

Datafile Sorted String Tables (SSTables). На відміну від журналу фіксації, дані записуються в цей файл асинхронно. SSTables періодично об'єднуються під час великих стиснень, щоб звільнити місце. Щоб зробити це, Cassandra з'єднує ключі, об'єднує стовпці і видаляти tombstones [1]. Операції читання можуть звернутися в кеш в пам'яті, і в цьому випадку не потрібно йти безпосередньо в data file на диску. Якщо можете дозволити Cassandra кілька гігабайти пам'яті, можна істотно підвищити продуктивність, коли кеш рядків і кеш ключів уражаються. [2]

Commit log періодично видаляються, після успішного точного додавання всіх її даних у виділені data file. З цієї причини commit log не будуть рости в будь-якому місці разом із розміром data file, так що диски не повинні бути великими. [4]

Concurrency. Cassandra відрізняється від багатьох сховищ даних в тому, що вона пропонує набагато більш високу продуктивність запису, ніж швидкість читання. Є два параметра, пов'язані з тим, як багато потоків може виконувати операції читання і запису: concurrent_reads і concurrent_writes. Загалом, за замовчуванням, що надаються Кассандри з коробки дуже добре. Але можете оновити concurrent_reads відразу налаштування перед запуском сервера. Це тому, що установка concurrent_reads є оптимальним при двох потоків на ядро процесора. За замовчуванням цей параметр 8, припускаючи, що чотирьох-ядерна коробка. Якщо використовується восьми-ядерний – налаштуйте його до 16. [1]

Установка concurrent_writes поводиться трохи інакше. Це повинна відповідати кількості клієнтів, які будуть писати одночасно на сервер. Якщо Cassandra є підтримка сервера веб-додатків, можете налаштувати цей параметр від значення за замовчуванням 32, щоб відповідати числу ниток сервер додатків доступні для підключення до Кассандру. Зазвичай в серверах додатків Java, таких як WebLogic воліють пули сполук бази даних розміром не більше 20 або 30, але якщо використовуєте кілька серверів додатків в кластері, ви повинні враховувати це.

Caching. Є кілька параметрів, пов'язані з кешуванням, як всередині Cassandra, так і на рівні операційної системи. Кеши можуть використовувати значну пам'ять, і це гарна ідея, щоб налаштувати їх ретельно. Є два основних кешей, вбудованих в Cassandra: кеш-рядки і кеш ключів. Кеш-рядки кешує повні рядки (всі їх стовпців), так що це є надбудовою кешу. Якщо використовуєте кеш рядків для даного сімейства стовпців, вам не потрібно буде використовувати кеш ключів на ньому.

Тому стратегія кешування повинна бути налаштована відповідно кільком факторам:

– використовувати тип кешу, який найкращим чином відповідає запитам;

– співвідношення розміру heapдо розміру кешу, і не можна дозволяти щоб кеш подавив heap;

– розмір рядків та розміром ключів. Зазвичай ключі будуть набагато менше, ніж цілі рядки.

Keys_cached це параметр, який вказує на кількість ключових місць - неключових значень - які будуть збережені в пам'яті. Це може бути визначено як дробове значення (число між 0 і 1) або у вигляді цілого числа. Якщо використовуєте частку, вказуєте, відсоток ключів в кеш, і ціле значення вказує на абсолютну кількість ключів, місце розташування яких буде зберігатися в кеші.

Ця установка буде споживати значну пам'ять, але може бути хорошим компромісом, якщо місцеположення не є жарким.

Метою disk_access_mode є забезпечення пам'яті mapped файлами, так що операційна система може читати кеш, тим самим знижуючи навантаження на внутрішні кеші Cassandra. Це звучить чудово, але на практиці, disk_access_mode є одним із менш корисних налаштувань, і в даний момент не працює точно так, як було спочатку передбачено. Це може бути покращено в майбутньому, але це так само, як ймовірно, що установка буде видалена. [3]

Також можете заповнити кеш рядків при запуску сервера. Щоб зробити це, використовуйте елемент preload_row_cache. Налаштування за замовчуванням є false, але якщо необхідно –встановіть його в true для підвищення продуктивності. Вартістю є те, що самонастроювання може зайняти більше часу, якщо є значні дані в Colum Family для попереднього завантаження.

Rows_cached параметр визначає кількість рядків, які будуть кешувати. За замовчуванням це значення дорівнює 0, що означає, що жодна рядок не буде в кеші, так що це гарна ідея, щоб включити кешування. Якщо використовуємо фракцію, вказуємо, відсоток від усього в кеш, і ціле значення вказує на абсолютне число рядків, місце розташування яких буде зберігатися в кеші. Хочете, щоб ретельно використовувати цей параметр, тим не менш, так як це може легко вийти з-під контролю. Якщо ColumFamily отримує набагато більше, ніж читає, пише, то установка цього числа дуже високим буде даремно витрачати значні ресурси сервера. Якщо ColumFamily має більш низьке відношення читає запис, але має рядки з великою кількістю даних в них (сотнях стовпців), то необхідно зробити деяку математику перед установкою цього числа дуже високо [1].

Durability. Довговічність (durability)– це властивість, яка записується, після завершення, зберігається завжди, навіть якщо сервер був убитий або

впав або втратив напругу. Це потребує виклику fsync сказати OS випередити його запис кеша на диск [7]. Простий спосіб забезпечити довговічність є fsync файлів даних з кожним записом, але це дуже повільно на практиці, тому що диск повинен випадково робити записи даних в місці розташування на фізичних дисках. Замість цього, як і інші сучасні системи, Cassandra забезпечує довговічність шляхом додавання запису в commit log першим. Це означає, що тільки commit log потрібно fsync'd, який, якщо commit log своєму власному об'ємі, позбуває необхідності в пошуках з commit log тільки додавання.

Конфігурація Cassandra за замовчуванням встановлює режим commit log_sync періодичні, змушуючи commit log бути синхронізованим кожні commit log_sync_period_in_ms мілісекунд, так що потенційно можемо втратити до того, що багато даних, якщо все репліки завершились аварійно протягом цього часу. Це поведінка за умовчанням є добре продуктивною навіть із commit log частками диска з каталогу даних. Також можемо вибрати пакетний режим, де Cassandra гарантуватиме, що вона синхронізується, перш ніж записати. Щоб уникнути синхронізації після кожного запису, групи Cassandra змінюються в партіях і синхронізуються кожні commit log_batch_ window_in_ms. При використанні цього режиму, рекомендується класти ваші commit log на окремо виділений пристрій [8, 9].

Висновки

Різноманітність налаштувань у файлі конфігурації допомагають налаштувати кластер Cassandra для підвищення продуктивності. Є кілька ізольованих налаштувань, які можна оновити в файлі конфігурації Cassandra.

Як правило, важливо відзначити, що просте додавання вузлів в кластер не поліпшить роботу самостійно. Необхідно реплецірувати дані належним чином, а потім відправити трафік на всі вузли з ваших клієнтів. Якщо не поширюєте клієнтські запити, нові вузли мережі можуть просто простоювати, коли кілька працюють постійно.

Список літератури

1. Eben Hewitt. Cassandra: The Definitive Guide. – USA: O'Reilly Media – 2010, - 330p.

2. NishantNeeraj. Mastering Apache Cassandra. Get comfortable with the fastest NoSQL database, its architecture, key programming patterns, infrastructure management, and more! . – Birmingham.: PACKT publishing, - 2014, - 340p.

3. RassellBradberry, Eric Lubow. Practical Cassandra. A Developer's Approach. – USA.: Addison-Wesley, - 2014, - 198p.

4. Robert Stricland. Cassandra High Availability. Harness the power of Apache Cassandra to build scalable, fault-tolerant, and readily available applications. – Birmingham.: PACKT publishing, - 2014, - 186p.

5. A Quick Introduction to the Cassandra Data Model [Электронныйресурс]. Режим доступа – http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/

6. Cassandra - таблицы с миллиардами столбцов [Электронный ресурс]. Режимдоступа – http://www.osp.ru/news/articles/2011/05/13006651/

7. Durability[Электронный ресурс]. Режим доступа – https://docs.datastax.com/en/cassandra/2.1/cassandra/dml/dml_durability_c.html

8. Durability[Электронный ресурс]. Режим доступа –https://wiki.apache.org/cassandra/Durability

9. Durability – Oracle vs Cassandra [Электронный-ресурс]. Режим доступа – https://lcmarques.com/2016/03/16/durability-oracle-vs-cassandra/

10. Как устроена apache cassandra [Электронный ресурс]. Режим доступа – http://habrahabr.ru/post/155115/

11. Погружение в СУБД Apache Cassandra [Электронный ресурс]. Режим доступа – http://www.ibm.com/developerworks/ru/library/os-apache-cassandra/

Надійшла до редколегії 17.03.2017

Рецензент: д-р техн. наук, проф. І.В. Рубан, Харківський національний університет радіоелектроніки, Харків.

АНАЛИЗ ФАКТОРОВ, ВЛИЯЮЩИХ НА ПРОИЗВОДИТЕЛЬНОСТЬ КЛАСТЕРА НЕРЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ CASSANDRA

Д.Д. Левченко

Главной идеей статьи является анализ факторов, влияющие на производительность кластера Cassandra. В работе была рассмотрены архитектура кластера нереляционной базы данных, проанализированы возможности такой архитектуры. В статье перечислены основные факторы, влияющие на производительность кластера. Особое внимание было уделено таким факторам как кэширование и долговечность.

***Ключевые слова**: база данных, Cassandra, архитектура, кластер, производительность, commit log, операции чтения, операции записи, кэш, кэширование, долговечность, fsync.*

ANALYSIS OF FACTORS AFFECTING PERFORMANCE CLUSTER NON-RELATIONAL DATABASE CASSANDRA

D.D. Levchenko

The main idea of the article is to analyze the factors that the performance of any living thing Cassandra cluster. The work has been viewed architecture cluster non-relational database analyzing possibility of such architecture. The article lists the main factors that affect the performance of the cluster. Particular attention was paid to factors such as caching and durability.

***Keywords**: database, Cassandra, architecture, cluster performance, commit log, read operation, write operation, cache, caching, durability, fsync.*