проблему дивергенції потоків і складності масштабування на великі матриці. Також виявлено затримки в процесі синхронізації даних між блоками. Наразі ведеться робота над дослідженнями оптимізації діагоналізації в плані обміну інформацією між блоками, а також пришвидшення за допомогою попереднього опрацювання матриці перед виконанням ітераційної частини.

Список джерел

1. Малашонок Г.І., Сухарський С.С., Алгоритм обчислення дводіагональної матриці ортогональним розкладанням на графічному процесорі. Наукові записки НаУКМА. Комп'ютерні науки. 2021. Том 4. doi: 10.18523/2617-3808.2021.4.10-15
2. Сухарський С. С., Алгоритм сингулярного розкладу на графічному процесорі. Проблеми програмування. 2023; 1: 30-37. doi: 10.15407/pp2023.01.030
3. S. Lahabar and P. J. Narayanan, "Singular value decomposition on GPU using CUDA," 2009 IEEE International Symposium on Parallel & Distributed Processing, 2009, pp. 1-10, doi:10.1109/IPDPS.2009.5161058.
4. Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki, "The Singular Value Decomposition: Anatomy of Optimizing an Algorithm for Extreme Scale", SIAM Review, v.60, 2018, pp. 808 - 865, doi: 10.1137/17M1117732.
5. Nishida, K., Nakasato, N., & Sedukin, V. "GPU-accelerated randomized singular value decomposition and its applications." Procedia Computer Science, vol. 29, pp. 2109-2119, 2014.

АРХІТЕКТУРА СИСТЕМИ УПРАВЛІННЯ БАЗАМИ ДАНИХ ДЛЯ ГРАНИЧНИХ ОБЧИСЛЕНЬ / DATABASE MANAGEMENT SYSTEM ARCHITECTURE FOR EDGE COMPUTING

Цьоменко Д.М. / Tsomenko D. M.

Київський національний університет імені Тараса Шевченка / Taras Shevchenko National University of Kyiv

Факультет комп'ютерних наук та кібернетики / Faculty of Computer Science and Cybernetics 03022, Україна, м. Київ, пр. Академіка Глушкова, 4д.

E-mail: denistsem@gmail.com, +380954030019

Анотація: This research examines the development of database management systems designed specifically for edge computing environments, incorporating local-first software principles to enhance functionality. The proposed DBMS architecture can process data close to its source by using edge computing, which minimizes latency, reduces the load on the network, and ensures real-time responses for analytics crucial for mission-critical applications. The local-first approach is primarily expressed with Conflict-Free Replicated Data Types, which enable efficient real-time collaboration across distributed environments while maintaining data consistency and resilience. This is especially beneficial for decentralized data processing and localized data storage.

1. Децентралізована обробка даних з використанням граничних обчислень.

З розвитком Інтернету речей та кількості пристроїв в ньому - обсяги даних зростають експоненціально, що призводить до навантаження на мережу та затримках в обробці даних в централізованих хмарних системах. Граничні обчислення вирішують цю задачу дозволяючи обробляти дані ближче до їх джерела зменшуючи затримки та збільшуючи пропускну здатність. Зміщення обробки та аналізу даних ближче до місця збору інформації відкриває можливості для аналізу в реальному часі, що критично для автономних систем, розумних міст, ситуаційної обізнаності на полі бою, авіації, тощо. Наприклад, самокеровані автомобілі потребують швидкої реакції на зібрану інформацію для запобігання зіткненням та адаптації до дорожніх умов. Запропонована архітектура пропонує встановлювати на крайні вузли спеціально розроблену базу даних та залишити більшість обробки на них. Вбудована база даних дозволяє кожному учаснику мережі збирати, обробляти, аналізувати дані незалежно. А принцип local-first дозволяє продовжувати роботу без постійного підключення до хмарного сервісу, таким чином

оптимізовуючи ресурси. Крім цього, децентралізована обробка збільшує стійкість і надійність загальної системи. Тому що навіть при втраті вузла кожен інший продовжує працювати незалежно, гарантуючи що система продовжує функціонувати. Застосування цієї моделі прибирає проблему з одиночною точкою відмови, яка типова для централізованих систем.

2. Безконфліктна синхронізація даних з CRDT

Архітектура системи баз даних для граничних обчислень включає інтеграцію безконфліктних реплікованих типів даних(CRDT). Це дозволяє досягти ефективнішу реплікацію даних та синхронізацію між декількома граничними вузлами та центральним сервером. Безконфліктні репліковані типи даних спроектовані для відслідковування змін незалежно на кожному вузлі та об'єднувати ці зміни без потреби в координації між вузлами. Використання CRDT в базах даних дозволяє уникнути проблем протоколів консенсусу і не потребує центрального серверу для вирішення конфліктів. Така автономія критична для граничних обчислень. Модулі синхронізації та реплікації керують розповсюдженням оновлень між вузлами та центральним сервером. Коли мережа відновлюється після деякого часу офлайн - цей підхід автоматично об'єднує різні стани для різних вузлів. Гарантується відсутність втрати даних і прозорість в вирішенні конфліктів між ними. Цей підхід використовувався для NoSQL баз даних таких як: Cosmos DB, Redis. Дана робота пропонує адаптувати і розширити його для реляційної моделі баз даних та об'єктної моделі сховища замість гібридної. Для цього використовується підхід який відслідковує операції, а не стан. Таким чином досягається підтримка комутативних реплікованих типів даних замість конвергентних реплікованих типів даних. Відсутність конфліктних типів поєднана з локальною обробкою даних дозволяє розробляти масштабовані рішення для керування великими обсягами даних у режимі реального часу.

3. Принцип local-first для покращення керування даними

Local-first програмне забезпечення це підхід до розробки, який передбачає, що дані зберігаються локально на пристроях користувачів і обробляються в першу чергу на місці, а не централізовано на сервері чи хмарі. Програмне забезпечення цього типу забезпечує локальний контроль, дозволяючи користувачам працювати офлайн, зберігаючи можливість синхронізації змін між різними пристроями, синхронізуючі їх тільки при наявності мережі. Для забезпечення цього принципу пропонується врахувати його в архітектурі бази даних. Використання запропонованої бази даних дозволяє іншим застосункам використовувати local-first підхід без значних змін в існуючі рішення. Головні принципами що виконуються є: локальне зберігання даних, автономність, асинхронна синхронізація змін, окреме резервне копіювання на кожному пристрої, що дозволяє системі автоматично створювати резервні копії та відновлювати дані, користувачі мають більший контроль над своїми даними, оскільки вони зберігаються локально і синхронізація відбувається лише за бажанням. Запропонована архітектура також відкриває можливість створювати повністю автономні офлайн системи, використовуючи тільки локальні API, що підвищує захищеність таких систем.

4. Безпека та конфіденційність через постквантове шифрування та контроль доступу на основі ролей (RBAC)

Важливою частиною архітектури баз даних є захищеність та контроль доступу. Для гарантування криптографічної стійкості в довгостроковій перспективі потрібно забезпечити захист не тільки від класичних атак, а й від злому за допомогою квантових алгоритмів. Були проаналізовані потенційні кандидати: Kyber, Saber, FrodoEKM, Dilithium, NTRU, Ієрархічний Діффі-Гелман з використанням суперсингулярних ізогеній. Критерієм вибору є не тільки захищеність, а також розмір ключів, швидкодія шифрування та дешифрування, сумісність з local-first архітектурою, підтримка ієрархічних ключів. Кінцевий вибором став алгоритм NTRU, який базується на задачі пошуку найближчого вектора в ґратках. Основною перевагою і ключовим фактором при виборі стала швидкодія та гнучкість. Алгоритм може бути налаштований на різні рівні безпеки, що дозволяє масштабувати його для різних задач та вимог та збалансувати безпеку та швидкість виконання залежно від способу використання бази даних. Адаптація цього алгоритму застосована для шифрування як окремих полів, так і цілих таблиць, а також захисту резервних копій. Контроль доступу на основі ролей (RBAC) регулює, хто може отримати доступ до даних, змінювати їх, виконувати запити, тощо. Підтримка ієрархії ролей робить налаштування гнучким та індивідуальним під кожну потребу користувача. Ця система спрощує управління доступом відкидаючи необхідність індивідуального налаштування для кожного користувача бази даних, покращує безпеку через призначення тільки тих дозволів, які потрібні користувачам для виконання їх функціональних обов'язків, дозволяє масштабувати систему шляхом додавання нових ролей.

Така архітектура дозволяє виконувати як і процесинг та аналіз даних у реальному часі в крайніх вузлах, забезпечуючи їх захищеність, стійкість та швидкодію. В наступних етапах

дослідження треба розробити архітектуру центрального серверу, а також проаналізувати та визначити які принципи та підходи використовувати при розробці ядра локальної бази даних. Включаючи: контроль паралельності та транзакцій, рівень ізоляції, типи зв'язків, логування, парсер, інтерфейс запитів, архітектуру, модель та організацію сховища, оптимізатор та виконувач запитів.

Список використаних джерел

1. An overview on edge computing research / K. Cao et al. IEEE access. 2020. Vol. 8. P. 85714–85728. URL: https://doi.org/10.1109/access.2020.2991734.
2. Consistent local-first software: enforcing safety and invariants for local-first applications / M. Köhler et al. IEEE transactions on software engineering. 2024. P. 1–12. URL: https://doi.org/10.1109/tse.2024.3477723.
3. Gao Z. Design and implementation of embedded database security and reliability. Journal of physics: conference series. 2020. Vol. 1550. P. 032034. URL: https://doi.org/10.1088/1742-6596/1550/3/032034.

АВТО СКЕЙЛИНГ ВЕБ ДОДАТКІВ НА AWS З ВИКОРИСТАННЯМ КОНТЕЙНЕРІВ ТА СЕРВЕРЛЕСС ТЕХНОЛОГІЙ/AUTO SCALING WEB APPLICATIONS IN AWS USING CONTAINERS AND SERVERLESS

Шевчук В.І./Shevchuk V.I.

Київський Національний Університет Імені Тараса Шевченка / Taras Shevchenko National University of Kyiv

03127, м. Київ, проспект Голосіївський 130/57, тел. +38 093 363 67 45,

E-mail: vitshev120@gmail.com

This paper considers three scenarios of auto scaling in a cloud environment. Investigated pros and cons of those. In particular, regarding stability and cost.

Авто Скейлинг є ключовою технологією для оптимізації ресурсів і забезпечення стабільності веб додатків. У цій роботі розглядається практичне впровадження авто скейлингу на прикладі трьох сценаріїв використання AWS ECS, SQS, CloudWatch та Lambda .

Перший сценарій охоплює авто скейлинг для виконання асинхронних складних задач. Використання Amazon SQS дозволяє розподілити навантаження між задачами. CloudWatch метрики, зокрема кількість не оброблених повідомлень у черзі, використовуються для автоматичного збільшення кількості ECS-контейнерів під час пікового навантаження та їх зменшення у періоди спаду. На рисунку 1 представлено приклад графіку завантаження черги SQS та кількості контейнерів синім та помаранчевим кольорами відповідно.

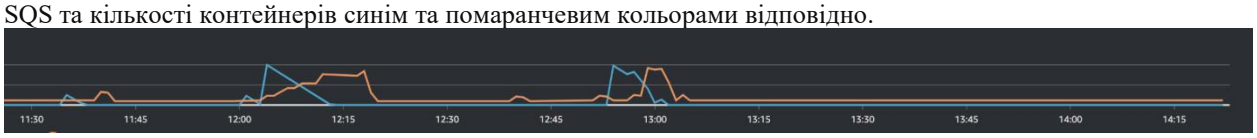

Рисунок 1. Графік завантаження черги SQS та кількості контейнерів

Другий сценарій демонструє авто скейлинг трейдинг бота для підтримки 10 активних користувачів на один контейнер. Контейнери автоматично додаються чи видаляються залежно від кількості запущених ботів для юзерів у реальному часі. Це рішення забезпечує стабільну роботу сервісу навіть при стрімкому зростанні активності користувачів, уникаючи перевантаження. Для роботи цього рішення було використано розподілення тасок у ECS по сервісам, з чітким завантаженням пам'яті та ресурсів на 10 контейнерів.

Третій сценарій стосується авто скейлингу API з використанням CloudWatch. Моніторинг таких метрик, як завантаження CPU та кількість запитів, дозволяє динамічно змінювати кількість ECS-контейнерів. Як альтернативу, можна було використати серверлес архітектуру (AWS Lambda).

Порівняємо ці рішення між собою. ECS із авто скейлингом демонструє кращу стабільність при високій інтенсивності запитів (>100 запитів на секунду), адже контейнери не потребують ініціалізації для кожного нового запиту. У той же час AWS Lambda забезпечує старт контейнерів в залежності від кількості запитів, ще може стати "вузьким місцем" при великих обсягах трафіку. ECS виграє за довготривалого навантаження через відсутність оплати за "cold