COMPUTER SCIENCE

АРХІТЕКТУРА МЕХАНІЗМІВ ОБРОБКИ ДАНИХ ТА СИНХРОНІЗАЦІЯ МОДУЛІВ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ SMART CITY

Войчик С. С., *бакалавр*
Тимошин Ю. А., *к. т. н, доцент*

Україна, м. Київ, Національний технічний університет України «Київський політехнічний інститут імені І. Сікорського», ФІОТ, кафедра Технічної Кібернетики

DOI: https://doi.org/ 10.31435/rsglobal_ws/31102018/6173

ARTICLE INFO

Received: 10 August 2018
Accepted: 21 October 2018
Published: 31 October 2018

KEYWORDS

IoT,
replication,
architecture,
synchronization,
databases,
smart city,
highly loaded systems.

ABSTRACT

The study is aimed to determine the main problem during developing and maintaining services of high load Smart City system and propose solutions for achieving eventual consistency between services. The main problem with the services and databases is a significant amount of requests which is produced by millions of devices and how to process and store it quite fast. Low latency, high scalability and failure resistance should be the main characteristic of the system. That is why choosing a database, a right strategy for database replicas, service synchronization and its monitoring are basic problems which must be solved first. There are several architecture and database types which are already used in more simple systems. Key aspect needs to be resolved – how to synchronize data between multiple services in Smart City system. To solve the problems we need to redevelop already existing technology which is used for more simple problems, join them and apply on new solution.

Citation: Войчик С. С., Тимошин Ю. А. (2018) Arkhitektura Mekhanizmiv Obrobky Danykh ta Synkhronizatsiia Moduliv u Vysokonavantazhenykh Systemakh Smart City. *World Science.* 10(38), Vol.1. doi: 10.31435/rsglobal_ws/31102018/6173

Вступ: У системах Smart City, навіть для невеликих міст, завжди буде оброблятись та зберігатись велика кількість даних. Основним джерелом даних є рухомі об’єкти: люди, приватні автомобілі, громадський транспорт. Крім того, дані агрегуються та обробляються різними сервісами вбудованими в систему, які в свою чергу також спілкуються між собою використовуючи визначені протоколи. За таких умов з’являються проблеми синхронізації, збереження даних, забезпечення відмовостійкості та прийнятного часу відповіді. В ідеалі, робота з спільними даними у розподіленій системі повинна виглядати так само, як у нерозподіленій системі - це означає, що все повинно виглядати так, ніби є лише одна копія даних, що читається та записується – без всяких реплік. Без сумніву сильна консистентність (strong consistency) [2] є найкращою моделлю узгодженості з точки зору програмістів додатків. На жаль, це не завжди можливо реалізувати в житті. Мережа повинна бути розділена, адже, якщо з якоїсь причини неможливе спілкування всіх вузлів, то у користувачів можуть бути

проблеми при використанні, оскільки вони будуть отримувати або застарілі дані, або не зможуть працювати з системою.

Результати досліджень: Кожне оновлення вимагає відправлень в обидва боки - до певної центрального вузлу або до певного кворуму серверів, а якщо комунікація відбувається повільно (наприклад, через географічну відстань між клієнтом і сервером або між репліками) то постраждає продуктивність і час відповіді між клієнтами системи. Всі компоненти розумного міста мають бути інтегровані за допомогою сервіс-орієнтованої архітектури. Міська архітектура є, по суті, масштабною розподіленою системою, яка по своїй суті є складною і децентралізованою. Різні платформи, неоднорідна обстановка та різноманітність мереж датчиків призведуть до проблем сумісності. Сервісно-орієнтована архітектура з її відкритими стандартами, такими як JSON, GraphQL, XML забезпечує не тільки взаємодію між різними платформами, але також підтримує модульний дизайн, повторне використання програмного забезпечення, взаємодію та інтеграцію додатків. Таку архітектуру легко впроваджувати, а деякі частини системи можна зробити на основі Event-Driven архітектури [3]. Для оброблення подій та ініціалізації веб-сервісів (для безпосередньої роботи) можуть бути використані три стратегії: проста, потокова і комплексна.

– Просте оброблення полягає в ініціалізації веб-сервісів по мірі реєстрації відповідних подій. Основна перевага такої системи — це робота в режимі реального часу. Мінус очевидний – не враховується фактор пріоритету.

– Потокове оброблення враховує існуючі залежності веб-сервісів і об’єднує кілька сервісів в один загальний потік. Цей підхід виправдовує себе в системах з великими накладними витратами на пошук і отримання інформації з баз даних.

– Комплексне оброблення – це так зване управління за відхиленнями. Будь-яка подія розцінюється як вихід системи зі стану рівноваги. Ініціалізація веб-сервісів переслідує єдину мету – повернути систему в стан рівноваги (задовольняючи потреби клієнтів системи).

Для забезпечення нормальної роботи систем є сенс вважати помилки при роботі систем не як рідкісні випадки, а як передбачувану частину нормальної роботи. Наприклад, зв'язок між мобільним клієнтом і сервером може вийти з ладу, оскільки користувач проїжджає через тунель або здійснює посадку на літак. Найчастіше, помилки роблять програму непридатною для використання, іноді не вказуючи на те, що пішло не так, і коли ми можемо очікувати нормального функціонування для відновлення. У гіршому випадку, збої можуть призвести до постійної невалідного стану даних або їх повної втрати.

Крім того, один з сервісів може оновлюватись в момент відправки запиту і через це бути певний час недоступним. В такому випадку можна приховати проблему від користувача, проганяючи всі важливі повідомлення від клієнтів через чергу.

Ці аспекти дуже важливі при розгляді того, як краще провести компроміс між стабільністю та доступністю. Проте на абстрактному рівні всі ці системи базуються на принципах узгодженості: спільні дані оновлюються в різних репліках, оновлення передаються асинхронно, а конфлікти постійно вирішуються.

Для реалізації асинхронних протоколів використовуються різні технології Message-Brokers (MB) [1]. Метою брокерів повідомлень є отримання вхідних повідомлень від додатків та виконання дій на них в майбутньому. У типовій архітектурі повідомлення, які вважаються бізнес-критичними, надсилаються в MB, де зберігаються повідомлення та розсилаються відповідним слухачам на надійний гарантований спосіб. Якщо в компоненті відбувається помилка обробки, перш ніж обробляти повідомлення, MB буде вимагати повідомлення після того, як компонент буде перезавантажено, або передасть іншому аналогічному серверу, який зможе впоратися з ним.

Для обробки подій, що можуть відправлятись на смарт пристрої можна застосовувати процесор подій. Цей компонент дозволяє розпізнавати послідовності повідомлень, які можуть сигналізувати тип події. Один тип послідовності може представляти загрозу безпеці. Інша послідовність може означати можливість щось продати комусь. Послідовність повідомлень має відбуватися протягом певного періоду часу. У світі IoT сервер CEP – (Complex Event Processing) може шукати послідовність повідомлень, які можуть вказувати на те, що хтось переміщується з кімнати в кімнату, щоб світло було включено або вимкнено, або навіть більш складні розрахування. Двигун CEP може зробити пристрої IoT розумними, визнаючи поведінку в деяких випадках і навіть в пристроях.

Історичні дані для IoT сервісів зростають швидко і часто досягають десятків петабайт. Популярним рішенням роботи з даними є зберігання історичних даних на репліках. У такому

разі, коли дані можуть зберігатись у декількох джерелах, потрібно використовувати map-reduce алгоритм. Крім того, історичні дані потрібно індексувати та використовувати шардінг для досягнення прийнятної швидкості запитів. Якщо сервіс буде використовувати хмарні технології, то важливу частину цих задач бере на себе постачальник послуг.

Для синхронізації БД, якщо, наприклад, потрібно побудувати навколо вже існуючого сховища пошуковий індекс, краще використовувати наступні техніки:

Використання невеликої програми, яка буде відслідковувати зміну даних в БД (update, delete) використовуючи певний стовбець (зазвичай Modified Date) та відправляти їх для оновлення в іншу БД [2]. Такий спосіб є підходящим для невеликих систем, де дані оновлюються не так часто і в більшості своїй є статичними.

Реплікація журналу (CDC – change data capturing): найшвидший спосіб - більш-менш золотий стандарт у реплікації даних. Вона включає в себе запит внутрішньої змінної журналу вашої бази даних кожні кілька секунд, копіювання змін у сховище даних та їх частому використанню. Всі зміни до вказаних вами таблиць і об'єктів завантажуються за замовчуванням, використовуючи журнал змін, тому нічого не втрачається. CDC - це не тільки швидший, надійніший спосіб, він також впливає значно менше на продуктивність бази даних під час запиту та допомагає уникати завантаження повторюваних подій. Однак, це потребує більше налаштувань і у випадку, якщо БД не підтримує його за замовчуванням, тоді доведеться самому використовувати допоміжне програмне забезпечення.

CDC - це найкращий спосіб для баз даних, які постійно оновлюються, він повністю підтримує видалення [2].

Так як всі процеси в системі потрібно контролювати для розуміння працездатності та помилок програмного забезпечення, потрібен моніторинг за бізнес - активністю сервісів. Цю роль можна покласти на сервіс моніторингу BAM (business activity monitor).

Він призначений для обробки великих потоків повідомлень з різних джерел, включаючи файли журналів або джерела, які, можливо, не орієнтовані на повідомлення. Сервер BAM [1] може обчислювати ключові бізнесові або операційні показники на основі всіх потоків повідомлень. Він може обчислювати показники SLA (Service Level Agreement), середні й підсумкові значення, а також генерувати події на основі значень, в діапазон яких потрапляли ці показники. Ця функціональність може використовуватися для цілей операцій або для бізнес-показників [1].

Часто потоки потрапляють у велику вітрину даних для подальшої обробки та аналітики. У середовищі IoT сервер BAM - це збирач даних для пристроїв IoT. Він передає дані в сховище даних, а також виконує обчислення та видає нову інформацію та події. Тут ви можете вказати, що ви хочете сумувати всю електроенергію, яка використовується у домі або в бізнесі, або в середньому за енергію протягом години. BAM може публікувати ці розрахунки періодично як події.

Висновки. Існує декілька типів архітектури та баз даних, які вже використовуються в більш простих системах, тому для побудови модулів Smart City з потрібними вимогами, доцільно реконструювати вже існуючі технології та об'єднати їх для застосування у модулях Smart City.

Основними вимогами до модулів та систем зберігання даних у високонавантажених системах Smart City є:

- Консистентність даних системи у часі між собою (Eventual Consistency) та можливість до швидких відновлень у разі падіння,
- Підтримання сховищами реплікацій журналу та механізмів обробки повідомлень,
- Взаємодія модулів у системі не напряму, а через посередника – наприклад, брокера повідомлень,
- Моніторинг активності сервісів, які використовуються, для розширення аналітики бізнес-процесів.

ЛІТЕРАТУРА

1. Архитектура корпоративных программных приложений – Мартин Фаулер. – ст.63-84, СПБ., 2006
2. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems 1st Edition by Martin Kleppmann 2017: p.151-270
3. Event-Driven Architecture: How SOA Enables the Real-Time Enterprise 2009: p.63-111