УДК 004.75

МІКРОСЕРВІСНА АРХІТЕКТУРА ТА ЇЇ ЗАДАЧІ НА ПРИКЛАДАХ З РЕАЛЬНОГО ЖИТТЯ

Дмитренко О.А.
Навчально-науковий інститут телекомунікаційних систем КПІ ім. Ігоря Сікорського, Україна
E-mail: olexandra.dmytrenko@gmail.com

MICROSERVICE ARCHITECTURE AND ITS TASKS ON EXAMPLES FROM THE REAL LIFE

Distributed systems and microservice architecture require in-depth understanding so that consensus problems do not arise during the process of approbation of the system but are solved at the stage of planning and technology selection. This article provides a description and comprehensible general comparison of microservice architecture with real-life problems. The "Two Generals' Problem" and "Byzantine Fault" issues with the possible solutions are described in the article. The presentation of the Paxus algorithm and ideas that go around it are also presented.

Розподілені системи та мікросервісна архітектура потребують досконального розуміння, аби задачі консенсусу не виникали в процесі апробації системи, але вирішувались на етапі планування та вибору технологій. У даній статті наведено опис та зрозумілі широкому загалу порівняння мікросервісної архітектури з задачами реального життя, та розглянуто задачі «двох генералів», «візантійських генералів». В тексті також описано можливі вирішення цих проблем, в тому числі, наведена ідея Паксус алгоритму.

Людський мозок - це загадка. Людина може дійти у своїх роздумах та уявленнях до того, що ніколи не існувало. Розум нам допомагає створювати нове та небувале, але в той же час, речі, які ми намагаємось створити, творяться за образом та подобою того, що вже існує. Наприклад, винаходячи літак, людина дивилась на птаха, роботи створюються щоб повторювати тіло людини та бути на нас схожими, машинний інтелект та нейронні мережі – щоб бути пародією на мозок людини та скомбінувати надможливості мозку у вигляді комп'ютерів із емоційною складовою, яка братиметься з даних про прийняття рішень реальних людей. У даній статті я описую мікросервісну архітектуру та ключові задачі консенсусу у вигляді порівнянь з усім відомими об'єктами та життєвими ситуаціями, щоб полегшити розуміння на перший погляд складних речей.

Мікросервісна архітектура – це багатоетапне виробництво. Зараз поширення набрала мікросервісна технологія, яка має під собою розподілену систему, або ж систему, що поділена на функціональні частини, які, зазвичай, можуть реплікуватись у випадку великого навантаження на них з метою ефективного реагування на кількість запитів, що необхідно обробляти. Раніше, натомість, застосунки розроблялись як моноліти, які обробляти стільки запитів, скільки обробляло їх найвужче місце. Також існує проміжний

варіант — множення одного монолітного сервісу на декілька додаткових серверів. Іншими словами, повноцінне програмне забезпечення існуватиме в декількох репліках. При організації рівномірного розподілення навантаження, це дозволить підвищити відмовостійкість застосунку, і, таким чином, пропускна можливість підвищиться. Мінусом даного підходу є менш ефективне використання ресурсів ніж у випадку мікросервісної архітектури, де, за потреби, збільшується тільки кількість необхідних елементів системи, а не створюється додаткова повноцінна система.

В сучасному світі, незабаром після того, як людство освоїло певну технологію, з'являються покращені версії цієї технології, або відбувається адаптація ідеї в інші сфери вжитку. Цікавим є те, що все нове насправді є старим і далеко не завжди забутим, те саме вже десь існувало, але в інших іпостасях, інші, і не обов'язково вчені, ці ж проблеми вже вирішував, тільки не завжди у тому самому середовищі.

Кажучи про винайдення мікросервісної архітектури, її ідею можна порівняти зі звичайним виробництвом, наприклад, сирним. В залежності від етапу виробництва, потрібна різна кількість ємностей та місця для пастеризації молока, відстоювання його після скисання та витримки сиру. Моноліт можна проасоціювати, з виробництвом, що має місце в одному приміщенні, де підтримується різна температура в залежності від етапу. Якщо етап витримки сиру займає багато часу, то виробництво чекатиме, доки даний етап не закінчиться.

Мікросервісною архітектурою було б створення різних кімнат, можливо, навіть, приміщень, для відповідних процесів з необхідною для них температурою та кількістю ємностей. Тоді б можна було в режимі реального часу приймати нове молоко та починати його обробляти. У випадку збільшення поголів'я корів, можна було б додатково розмістити більшу кількість ємностей в одні кімнати, а якщо розміри не дозволяються, побудувати нові зали, призначені для певного процесу. Відзначу, що зважаючи на об'єми різного роду сировини, додаткові приміщення можуть знадобитись для скисання молока, адже пастеризація проходить швидше, ніж скисання, та існуючих потужностей може вистачити. Сир не займає багато місця, а отже і для його зціджування можливо вийде обійтись лише додатковими стелажами. Якщо кількість вхідного молока зменшуватиметься, додаткові кімнати та ємності можна не заповнювати сировиною, а отже і не витрачати додаткові кошти на підтримку робочого стану. Словами мікросервісної архітектури, ми збільшуємо чи зменшуємо кількість робочих серверів, виділених на певний мікросервіс, або ж залучаємо додаткові потоки для обробки вхідних запитів.

Задача гарантованої комунікації або двох генералів. Добре, коли новий процес працює, полегшує життя та робить його якіснішим. Але ж зазвичай, з ускладненням самого виробництва, необхідно створювати нові та додаткові процеси. Наприклад, розташувавши різні частини виробництва в різних приміщеннях, необхідно організувати транспорт між приміщеннями, що вимагатиме не тільки додаткових вкладень, але і ставитиме під загрозу саме виробництво, у випадку неможливості транспортування через

відсутність бензину, електрики, поламки транспортних засобів, сторонніх подій, що заважатимуть руху, тощо. Проводячи аналогію з мікросервісною архітектурою, потрібно організувати відправлення повідомлень між частинами застосування. Це, в свою чергу, створює питання доставки сповіщень, а також їх правдивість.

Проблема транспортування сповіщень між елементами розподіленої системи опублікована ще у 1975р у [0] та отримала назву «Задача двох генералів» у 1978р. Суть полягає в тому, що два генерали атакують спільного ворога, армія якого розташована між арміями спільників. Успіх буде тільки у випадку одночасної атаки, про яку слід домовитись. Головний генерал має надіслати іншому сповіщення про атаку, а також отримати від нього підтвердження. Далі можливі дві негативні ситуації: до другого генерала не доїде гінець, або ж сповіщення буде отримано, але перший генерал не отримає підтвердження через перехоплення гінця на зворотному шляху. Таким чином, неможливо гарантувати, що обидві армії підуть в атаку на ворога в один і той самий час.

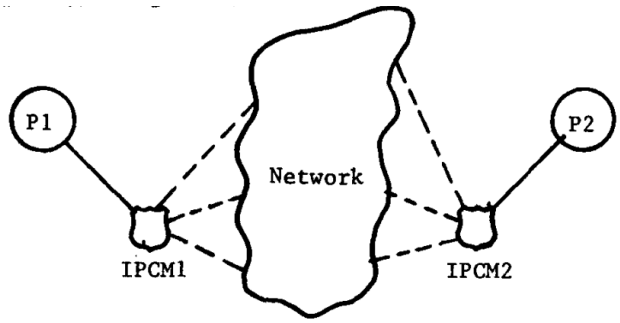

Рис. 1. Задача двох генералів у розподіленій мережі.

Вище наведено Рис. *1* з [0], що описує наведену проблему мовою мікросервісної архітектури. P1 (partition 1) та P2 (partition 2) – це окремі мікросервіси, між якими має відбутись комунікація. IPCM – inter process communication mechanism, тобто механізм кожної сторони, через який відбувається комунікація, наприклад, REST протокол. Network – мережа, яка з одного боку служить шляхом комунікування, а з іншого, – може порватись в любий момент, тому асоціюється з ворожою армією.

Задача гарантування отримання відповіді відправником не має розв’язку. Тим не менш, ситуація достатньо поширена тому нижче я наводжу способи суттєво зменшити вірогідність неотримання відповіді.

1) Побудувати безпечний канал зв’язку між двома арміями, але це додаткові ресурси.

2) Замість надсилання одного гінця надіслати багато, наприклад, 100. Інформацію вважати прийнятою у випадку отримання однієї чи декількох відповідей. Декілька відповідей може знадобитись коли сповіщення може бути підроблене по дорозі.

3) Заради оптимізації ресурсів, можна відправляти кожного наступного гінця після того, як пройшов час, за який мав повернутись перший.

Додатково, щоб зрозуміти якість каналу, можна нумерувати сповіщення та робити висновки з отриманих результатів.

Задача неправдивих сповіщень або візантійських генералів. Окрім питання доставлення сповіщень, постає питання правдивості отриманих сповіщень. Якщо зв’язок

погано захищений, то по дорозі можна підробити сповіщення. Тим не менш, така проблема може статись навіть при захищеному з'єднанні, коли в одного мікросервіса чи розподіленої бази даних з різних причин застаріла інформація, яка вже неактуальна. Тобто, необхідно гарантувати, що або ж інформація буде останньою, або даний вузол системи не братиме учать в консенсусному прийнятті рішення.

Проводячи паралель з реальним життям – можемо уявити армію, що складається з багатьох легіонів. Головний генерал, який може бути зрадником, має дати наказ генералам легіонів або нападати або відступати. Зважаючи на те, що субординати можуть теж бути зрадниками, і їх може бути до третини, успіх буде тільки у випадку нападу всіма чесними генералами. Таким чином, накази головнокомандуючого слід перевіряти, а без комунікації війна буде програною. Дана задача називається «Задачею візантійських генералів» і також не має розв'язку. Така проблематика була описана у 1985 році у [0] та ставила під питання невалідність тільки одного вузла системи. Тим не менш, схожу задачу розглядали ще з 1950х років у NASA та у авіаційній галузі з метою створення бортових комп'ютерів, які б мали

- декілька реплікацій даних та окремих обчислювальних машин, а також,
- паралельні процеси однакових обчислень, які б гарантували правильність висновків та максимально унеможливлювали помилку,
- шукали та знаходили помилку методом порівняння результатів та голосуванням (більшість однакових результатів вказує, що це правильний результат)
- виправляли помилку,
- реконфігурувались щоб позбавитись зламаних вузлів та не брати їх «думку» до уваги [0].

У наведеній статті [0] використовувалось шість реплікацій для забезпечення коректної роботи тогочасних літаків, які обмінюються станом між собою кожні 50мс, але в загальному випадку реплікацій може й не потрібно стільки. Наприклад, в Європе́йській організації з я́дерних досл́іджень, що знаходиться в Женеві, використовується три репліки даних, які розташовані в трьох різних країнах на випадок війни та бомбардувань. Варто зазначити, що кількість даних величезна, адже йдуть записи щомоментного розташування частинок в адронному колайдері.

Паксос протокол та модель частково працюючого парламенту. Існує не один алгоритм, що допомагає вирішенню проблеми візантійських генералів. Я хочу навести алгоритм, який вирішуючи проблему нерегулярності роботи окремих вузлів, також, вирішував проблему оповіщення новими законами депутатів, які час від часу виходили з зали засідань. Стаття, що описує даний алгоритм була надрукована у 1989 Леслі Лампортом, але до неї поставились серйозно пізніше, коли була надрукована стаття без порівняння процесу з грецьким парламентом на острові Паксос.

Ідея алгоритму полягає у тому, що за допомогою порівняння поточного стану системи та метаінформації про дані на етапі їх зчитування з різних реплік, шляхом голосування приймається рішення, які саме дані валідні, якщо вони не співпадають.

Нижче я опишу реалізацію запису та читання інформації за Паксос алгоритмом.

Існує три ролі:

- Пропонувач – пропонує закони або ж висловлює бажання запису певної інформації, нумеруючи її у зростаючій послідовності.
- Приймачі – депутати, які розглядають нові закони, або ж вузли системи, які отримують сповіщення. Розгляд відбувається тільки у випадку, якщо є кворум (більшість працюючих вузлів системи).
- Секретар (учень) – який, бере-запиує виключно прийняті закони, та віддає їх кінцевому користувачу [0].

Метод, через який відбувається голосування заснований на порівнянні максимального номеру вже існуючого запису в пам'яті вузла. Тобто, якщо новий номер більший за вже існуючий, приймач відповідає обіцянкою не розглядати сповіщення з меншим номером, а якщо ні – приймач відхиляє запис та у відповідь надсилає максимальний наявний в нього номер. Якщо більшість приймачів відповіла обіцянками прийняти сповіщення, воно буде розіслане з запитом «прийняти» усім наявним вузлам, і вони мають його прийняти, якщо попередньо не відповіли, що мають вищий номер. Сповіщення відсилається клієнтам тільки після того, як воно було прийняте кворумом [0].

Висновки. З кожним днем кількість розподілених систем стає все більше й більше, адже бізнесмени хочуть бути впевненими, що їх продукт витримає любе навантаження та залишиться працювати навіть у випадку бомбордувань в місці з частиною серверів. Питання вирішення конфліктних ситуацій у розподілених системах залишається актуальним вже 70 років, адже з розвитком інформаційних технологій з'являються нові нюанси, ідеї та потреби. Зважаючи на те, що навчання нового покоління, а також розвиток професіоналів потребують вигадливості та простоти, аби бути почутим та зрозумілим, я навела деякі сучасні задачі та побудови розподілених систем, які походять ще з давнини, та запропонувала їх розв'язки.

Література

1. Akkoyunlu, Eralp A., K. Ekanandham and Richard Huber. "Some constraints and tradeoffs in the design of network communications." // presented at the Proceedings of the fifth ACM symposium on Operating systems principles, SOSP '75, 1975, 01 листопада, ст. 67-74, DOI:10.1145/800213.806523
2. Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. J. ACM 32, Volume 32, Issue 2, 1985 квітень, 374–382, DOI:10.1145/3149.214121
3. Goldberg, Jack. "The SIFT computer and its development.". Digital Avionics Systems Conference, 1981. ст.285-289 DOI:10.2514/6.1981-2278.
4. Kevin Babitz. "The Strange Story of the Paxos Algorithm". 2021, 18 лютого. [Онлайн] https://towardsdatascience.com/the-strange-story-of-the-paxos-algorithm-52a9f3f53ae0
5. Leslie Lamport. 1998. The part-time parliament. ACM Trans. Comput. Syst. 16, 2 (May 1998), 133–169, DOI:10.1145/279227.279229.