УДК 004.75,004.4’2
DOI: 10.18523/2617-3808.2024.7.58-62

Давиденко А. М.

ЗАБЕЗПЕЧЕННЯ ПОРЯДКУ ОБРОБЛЕННЯ ПОВІДОМЛЕНЬ У РОЗПОДІЛЕНИХ СИСТЕМАХ

У статті проаналізовано основні виклики, які є актуальними для розробників розподілених систем, зокрема стосовно забезпечення порядку оброблень подій, що відбуваються у системі, та його впливу на загальний стан системи. Відповідно до цього, запропоновано класифікацію розподілених систем. Наведено приклади використання можливостей сучасних брокерів повідомлень RabbitMQ та Apache Kafka, які дають змогу забезпечити оброблення повідомлень лише одним споживачем у розподіленій системі, та порівняно їхні можливості.

Ключові слова: розподілені системи, програмування, алгоритми, класифікація, брокери повідомлень.

Вступ

У розподіленій системі (РС) комп’ютери (вузли або процесори) з’єднані та взаємодіють один з одним через комунікаційну мережу. На відміну від централізованої системи, де весь процес оброблення даних керується одним комп’ютером або сервером, у РС завдання і дані розподіляються між декількома вузлами. Забезпечуючи масштабованість і стійкість, РС вносять нові виклики — неминучий паралелізм і складність впорядкованих комунікацій між вузлами всередині системи. Тому проблема підтримки впорядкованості повідомлень (також відомих як події) є актуальною і широко досліджується.

У цій статті проаналізовано актуальні виклики у розробленні РС і наведено типові сценарії, за яких виникає необхідність упорядкування повідомлень для їх оброблення, а також запропоновано класифікацію РС відповідно до вимог щодо збереження поточного стану та обчислень.

1. Виклики у розробленні розподілених систем

Лампорт у своїй статті вводить часову логіку відношення під назвою «відбулося раніше», яке допомагає описати концепцію часткового впорядкування (логічний годинник) [3]. Це поняття далі використовують для опису розподіленого алгоритму, спрямованого на визначення часткового порядку подій у системі, а також проблем із синхронізацією. Крім того, Лампорт уперше формалізував важливе поняття причиновості (causality) в контексті РС. Підхід до визначення порядку подій, який він запропонував, базується на часових мітках. Кожному повідомленню в джерелі надають часову мітку, яку використовують для доставлення і оброблення повідомлення в правильному порядку. Безсумнівно, ця робота заклала основу для впорядкування повідомлень у РС і дала поштовх для багатьох інших досліджень у контексті РС.

Нині існує велике різноманіття запропонованих алгоритмів, які забезпечують контрольований порядок доставлення повідомлень, однак кожен з них має свої переваги і недоліки, що вкотре підкреслює актуальність цієї проблеми [1].

Попри те, що проблеми РС уже досить добре досліджені, певні питання досі залишаються без відповіді [7]. Наприклад, чи має впорядкування повідомлень бути вирішено за допомогою проміжного програмного забезпечення (middleware) або все ж таки кожним прикладним програмним забезпеченням? Порядок оброблення повідомлень є критично важливим для коректного стану РС. Часто проблеми оброблення повідомлень не помічають на етапі розроблення через недостатнє тестування або незначні навантаження. Проблема довгий час може лишатися непоміченою, якщо час оброблення події є меншим за час між тим, як нові події стаються у системі. Саме тому це важливо враховувати на етапі проєктування РС.

2. Класифікація розподілених систем

Ми пропонуємо класифікувати розподілені системи (РС) за такими ознаками стосовно вимог щодо поточного стану системи.

Перший клас — це РС, у яких необхідно забезпечити коректність поточного стану. В цьому разі можливе паралельне оброблення повідомлень системами, адже послідовне оброблення повідомлень не є критичним. Однак необхідно відстежувати порядок повідомлень задля збереження наміру повідомлень. Розглянемо приклад, у якому система може мати три стани: *Off*, *Initalizing* та *On*. Припустимо, що процес p_1 надсилає подію E_1 = *Initalizing* і подію E_2 = *On* із мінімальною часовою різницею. У РС може виникнути ситуація, коли подію E_2 буде оброблено першою, а подію E_1 — другою. В такому разі, подію E_1 може бути відкинуто програмною логікою під час її оброблення як неактуальну. Використовуючи лінійні темпоральні логіки [4], очікуваний стан системи можна представити як F (*G On*). Тоді порядок повідомлень можливо встановити, використовуючи логічний годинник, наприклад годинник Лампорта або векторний годинник.

Другий клас — це РС, у яких наявні причиново-наслідкові залежності відповідно до поточного стану системи. В цьому разі необхідно забезпечити строгий порядок оброблення кожного повідомлення. Розглядаючи приклад із попереднього пункту, припустимо, що у РС існують умови на перехід із попереднього стану у наступний (наприклад, *Off* → *Initalizing*). Пропустивши стан *Initalizing*, як у першому прикладі, ми можемо порушити логіку роботи РС. Враховуючи ці вимоги, систему можна представити як G (*Off* ⋃ (*Initalizing* ⋃ *G On*). У цьому випадку навіть у разі масштабування сервісів необхідно передбачити оброблення повідомлень лише одним споживачем в один момент часу. Деякі брокери повідомлень мають вбудовані для цього засоби (як-от RabbitMQ чи Apache Kafka). Варто зауважити, що в разі застосування цього підходу найбільше обмежується пропускна здатність системи.

Третій клас — це РС, у яких наявні причиново-наслідкові залежності відповідно до поточного стану системи в рамках певного контексту. Цей варіант накладає ті самі обмеження, що й попередній, але вони є незалежними для кожного контексту. Завдяки цій умові можливо покращити пропускну здатність системи. Тому цей варіант підходить для тих систем, в яких зберігається поточний стан і можливо виділити об'єкти у стані системи, які є незалежними в обробленні. В такому разі одночасне оброблення подій для кількох незалежних об'єктів не вплине на коректний стан системи.

За допомогою такої класифікації на етапі проєктування розподіленої системи можливо визначити необхідний технічний підхід для взаємодії між компонентами РС відповідно до поставлених вимог.

3. Забезпечення впорядкованого оброблення повідомлень за допомогою брокерів повідомлень

Брокери повідомлень — це проміжне програмне забезпечення, яке дає змогу різним системам, службам і програмам обмінюватися інформацією та взаємодіяти між собою. Простіше кажучи, брокери повідомлень діють як посередники для роботи різних програмних систем, наприклад вебдодатків. Окрім базових функцій, брокери повідомлень за допомогою вбудованих механізмів дають можливість досягти бажаних властивостей системи, як-от забезпечити впорядковане оброблення повідомлень.

3.1. RabbitMQ

RabbitMQ — це брокер повідомлень, який імплементує систему асинхронного обміну повідомленнями між компонентами програмної системи через відкритий стандарт протоколу AMQP (Advanced Message Queuing Protocol). За допомогою RabbitMQ виробники публікують повідомлення до обмінників (exchanges), які потім направляють це повідомлення до підв'язаних до нього черг. Черга є основним компонентом RabbitMQ, який зберігає повідомлення доти, поки їх не буде доставлено одержувачам. Черга є абстракцією, де повідомлення надходять після їх публікації від виробників і чекають на оброблення споживачами. Важливо, що черга дає змогу обробляти повідомлення незалежно від часу їх надсилання та отримання, що дозволяє розподіляти навантаження між кількома споживачами.

RabbitMQ дає можливість використовувати властивість SAC (Single Active Consumer) для певної черги та є корисною, якщо повідомлення мають бути оброблені у тому самому порядку, у якому вони

надійшли до черги [5]. SAC дозволяє мати лише одного активного споживача повідомлень з черги і перемикатися на іншого доступного у разі, якщо активний споживач виходить із ладу. Щоб покращити пропускну здатність системи та забезпечити послідовне оброблення повідомлень у рамках контексту, можна використовувати плагіни, що підтримують механізм узгодженого хешування (consistent hashing) [6]. Слід зазначити, що проблема впорядкованого оброблення повідомлень може виникнути в разі створення не тільки РС, а й багатопотокових додатків.

3.2. Apache Kafka

Apache Kafka — це розподілене сховище подій і платформа для їх багатопотокового оброблення з високою пропускною здатністю та низькою затримкою, яке базується на власному протоколі. На відміну від RabbitMQ, Kafka основується на моделі витягування повідомлень (pull-based). Основою Kafka є розподілений журнал (log). Kafka використовує концепцію тем (topics) і розділів (partitions). Якщо порівнювати з RabbitMQ, тему можна уявити як чергу, що поділяється на розділи, базуючись на ключі повідомлення. Наприклад, у прикладній системі таким розділом може бути назва країни. Виробник додає своє повідомлення до теми, а споживач самостійно вичитує повідомлення, оновлюючи зміщення (offset), що фіксує індекс останнього прочитаного повідомлення. Таким чином, повідомлення зберігаються до планового очищення, що дає змогу за необхідності перечитувати повідомлення кілька разів, а брокер повідомлень не відстежує вичитування повідомлень споживачами.

Задля досягнення впорядкованого оброблення повідомлень Apache Kafka надає вбудовані засоби. Наприклад, щоб забезпечити строгий порядок оброблення повідомлень у рамках теми, необхідно використовувати єдиний розділ для всіх повідомлень і одну групу споживачів (consumer group) [2]. За ідеологією Kafka один розділ може мати лише одного споживача в рамках кожної групи споживачів (рис. 1). При цьому в одній групі споживачів може бути більша кількість споживачів, аніж розділів. У такому разі не призначені до розділів споживачі будуть неактивними, і у разі виходу з ладу одного з активних споживачів іншого буде під'єднано автоматично. Щоб досягти послідовного оброблення повідомлень у рамках контексту, тему може бути поділено на розділи. При цьому послідовне оброблення повідомлень здійснюватиметься в рамках кожного з розділів.

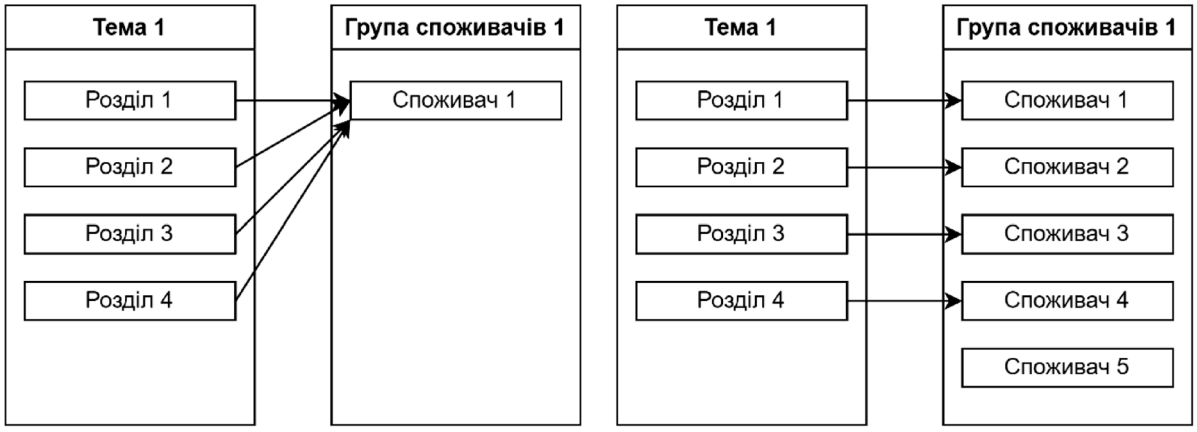

Рис. 1. Можливі сценарії розподілу розділів і споживачів у Apache Kafka: *1* — один споживач обробляє повідомлення з кількох розділів; *2* — один зі споживачів не є активним, інші споживачі підключені до розділів у відношенні 1:1

Таблиця 1

Застосування брокерів повідомлень до класифікованих РС

Брокер / Клас РС	RabbitMQ	Apache Kafka
Клас 1	Логічний годинник	Логічний годинник
Клас 2	SAC	Єдиний розділ у темі, одна група споживачів
Клас 3	SAC + плагін з узгодженим хешуванням	Окремі розділи у темі, одна група споживачів

Отже, сучасні брокери повідомлень надають ефективні засоби, що дають змогу забезпечити контрольований порядок оброблення повідомлень для РС другого і третього класу. Apache Kafka надає ці засоби «з коробки», а для деяких сценаріїв RabbitMQ потрібно встановити додаткові плагіни. Для підтримки РС першого класу брокери не надають вбудованих засобів. Це можна пояснити необхідністю тісної інтеграції з бізнес-логікою прикладного програмного забезпечення, адже рішення про відкидання повідомлення чи події може бути здійснено лише на рівні прикладного програмного забезпечення.

Висновки

У рамках цієї роботи висвітлено деякі актуальні виклики, з якими стикаються розробники РС. Незважаючи на те, що зазначені проблеми вже доволі добре досліджені, підходи до їх вирішення досі не є уніфікованими.

У статті запропоновано класифікацію РС відповідно до бажаних характеристик стосовно збереження поточного стану. Наведено приклади використання засобів сучасних брокерів повідомлень RabbitMQ та Apache Kafka, які дають змогу забезпечити оброблення повідомлень лише одним споживачем, порівняно можливості їх застосування до класифікованих РС.

Подальшим розвитком цієї роботи є створення дизайн-патернів для описаних типів РС. Також цікавим може бути застосування елементів штучного інтелекту для визначення контексту поточного стану систем і визначення подій, які не впливають на нього задля можливості паралельного оброблення повідомлень без порушення стану системи.

Список літератури

1. Défago X. Total order broadcast and multicast algorithms / X. Défago, A. Schiper, P. Urbán // ACM Computing Surveys. — 2004. — Vol. 36, no. 4. — Pp. 372–421. — https://doi.org/10.1145/1041680.1041682.
2. Kafka 2.0 Documentation [Electronic resource]. Apache Kafka. — Mode of access: https://kafka.apache.org/20/documentation.html (date of access: 22.09.2024). — Title from screen.
3. Lamport L. Time, clocks, and the ordering of events in a distributed system / L. Lamport // Communications of the ACM. — 1978. — Vol. 21, no. 7. — Pp. 558–565. — https://doi.org/10.1145/359545.359563.
4. Pnueli A. The temporal logic of programs / A. Pnueli // Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS). — 1977. — Pp. 46–57. — https://doi.org/10.1109/SFCS.1977.32.
5. RabbitMQ Documentation [Electronic resource]. *RabbitMQ: One broker to queue them all | RabbitMQ*. — Mode of access: https://www.rabbitmq.com/docs (date of access: 22.09.2024). — Title from screen.
6. Vahab M. Consistent hashing with bounded loads / M. Vahab, M. Thorup, M. Zadimoghaddam // Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. — 2017. — Pp. 587–604. — https://doi.org/10.1137/1.9781611975031.39.
7. van Steen M. Distributed Systems 4th edition [Electronic resource] / M. van Steen, A. S. Tanenbaum // DISTRIBUTED-SYSTEMS.NET. — Mode of access: https://www.distributed-systems.net/index.php/books/ds4/ (date of access: 22.09.2024). — Title from screen.

References

Défago, X., Schiper, A., & Urbán, P. (2004). Total order broadcast and multicast algorithms. *ACM Computing Surveys*, *36* (4), 372–421. https://doi.org/10.1145/1041680.1041682.

Kafka 2.0 Documentation. (n.d.). Apache Kafka. https://kafka.apache.org/20/documentation.html.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, *21* (7), 558–565. https://doi.org/10.1145/359545.359563.

Pnueli, A. (1977). The temporal logic of programs. *Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS)*, 46–57. https://doi.org/10.1109/SFCS.1977.32.

RabbitMQ Documentation. (n.d.). RabbitMQ: One broker to queue them all | RabbitMQ. https://www.rabbitmq.com/docs.

Vahab, M., Thorup, M., & Zadimoghaddam, M. (2017). Consistent hashing with bounded loads. *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 587–604. https://doi.org/10.1137/1.9781611975031.39.

van Steen, M., & Tanenbaum, A. S. (n.d.). *Distributed Systems 4th edition*. DISTRIBUTED-SYSTEMS.NET. https://www.distributed-systems.net/index.php/books/ds4/.

A. Davydenko

ENSURING THE ORDER OF MESSAGE PROCESSING IN DISTRIBUTED SYSTEMS

In a distributed system, computers, also known as nodes or processors, are connected and communicate with each other through a communication network. Unlike a centralized system, where a single computer or server handles all processing tasks, in a distributed system, tasks and data are distributed among

multiple nodes. While providing great scalability and resilience, distributed systems introduce new challenges – unavoidable concurrency and the difficulty of maintaining ordered communication between nodes within a system. That is why distributed systems and applications are especially difficult to create and maintain. Since the problem of preserving message ordering (also known as event ordering) is crucial, it has been widely studied. Some of the key findings are covered in this paper.

Many proposed algorithms ensure a strict order of message delivery, but each has its advantages and disadvantages, which once again emphasizes the urgency of this problem.

This paper highlights some of the current challenges faced by developers of distributed systems. Despite the fact that these problems are already well-researched, approaches to their resolution are still not settled.

The article proposes a classification of distributed systems based on the desired characteristics in terms of maintaining and evaluating the current state. An example of using modern message broker tools (such as RabbitMQ and Apache Kafka), which ensure the processing of messages by only one consumer, is given.

Further development of this work involves creating design patterns for the described types of distributed systems. Additionally, incorporating artificial intelligence elements to determine the context of the current state of systems and identify events that do not affect it could enable parallel processing of messages without disrupting the system state.

Keywords: distributed systems, programming, algorithms, classification, message brokers.

Матеріал надійшов 27.09.2024