УДК 004.421
DOI https://doi.org/10.32838/2663-5941/2019.6-1/32

Шайдурова К.А.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ГАРАНТОВАНИЙ ПОРЯДОК ДОСТАВКИ ПОВІДОМЛЕНЬ У ХМАРНИХ СИСТЕМАХ

Однією з найважливіших абстракцій для хмарних розподілених систем є консенсус – узгодженість між усіма вузлами щодо певного питання. Досягнення консенсусу є провідним для гарантування порядку доставки повідомлень під час асинхронної обробки. Під час досягнення консенсусу додатки можуть використовувати його для різних цілей – процесу реплікації, гарантії відмовостійкості вузлів, а також захисту від втрати даних. Щоб забезпечити роботу хмарних розподілених систем потрібно допустити можливість часткових відмов і вбудувати в програмне забезпечення механізми забезпечення відмовостійкості наряду зі збереженням порядку доставки. Будь-які відмови або затримки у роботі вузлів можуть призвести до втрати повідомлень та, відповідно, до порушення порядку їх отримання та обробки.

Гарантований порядок доставки є важливим для випадків, коли швидкість обробки повідомлень є менш критичною, ніж обробка даних в тому порядку, в якому вони надходили від споживача, наприклад, в банківських операціях, аукціонах. У цій статті проведено аналіз наявних алгоритмів консенсусу та вимог, що ставляться до них. Також розкрито роботу брокера повідомлень, що гарантує порядок доставки повідомлень з використанням алгоритму консенсусу Raft. У статті викладено опис механізму вибору лідера серед вузлів видавців та споживачів для забезпечення реплікації записів та можливості продовжити роботу через перевибори в разі, якщо поточний лідер перестав надсилати періодичне серцебиття. Також у статті наведено механізм зміни складу кластера, який передбачає використання проміжної конфігурації, яка має бути записана кворумом вузлів перед тим, як буде зроблено запис нової конфігурації, що запобігає обранню двох лідерів, які функціонують одночасно. Описана запобіжна перевірка, що не дає можливості відключеним вузлам постійно ініціювати нові вибори, тим самим сповільнюючи роботу системи.

Ключові слова: *консенсус, загальний порядок передачі, відмовостійкість, брокер повідомлень, реплікація, Raft.*

Постановка проблеми. Гарантування порядку доставки повідомлень у хмарних системах стикається з проблемою неможливості досягнення консенсусу в асинхронних розподілених системах. Якщо можливий збій хоча б одного процесу в системі, проблема досягнення консенсусу не має детерміністичного рішення. Для практичного застосування дотримання всіх вимог консенсусу зазвичай не є критичним або може бути компенсоване додатковими запобіжними механізмами системи. Тому для забезпечення гарантування порядку повідомлень в брокерах повідомлень можна застосувати алгоритми з менш жорсткими вимогами до консенсусу.

Аналіз останніх досліджень і публікацій. У галузі розподілених обчислень є «Проблема загального порядку передачі». Загальний порядок передачі передбачає, що повідомлення повинні бути доставлені всім учасникам у будь-якому порядку, доки він однаковий для всіх. Доставка повідомлень FIFO в розподіленій системі – це власне окремий випадок проблеми із додатковим обмеженням, накладеним на передачу повідомлень [7].

Проблема загального порядку передачі еквівалентна проблемі досягнення розподіленого консенсусу. Проблема консенсусу визначається таким чином: кожен процес пропонує початкове значення для інших, і, незважаючи на збої, всі функціонуючі процеси повинні узгодити спільне значення, яке повинно бути одним із запропонованих. Ця проблема не має детерміністичного рішення в асинхронних розподілених системах, в яких можливий збій хоча б одного процесу – так званий результат неможливості Фішера-Лінча-Патерсона (FLP) [2]. По суті, неможливість досягнення консенсусу та загального порядку передачі випливає із притаманних асинхронним системам труднощів визначити, чи дійсно процес вийшов з ладу чи є лише дуже повільним.

Результати FLP дослідження свідчать про те, що на практиці цю проблему можна вирішити,

лише уточнюючи вимоги до досягнення консенсусу. Рішення конкретних практичних задач зазвичай потребує вдосконалення моделей розподілених обчислень, які відобразили б реалістичні припущення щодо часу відгуку процесора та часу зв'язку, а також формулювання менш жорстких вимог щодо вирішення проблем досягнення консенсусу та загального порядку передачі [3].

Наприклад, вимога щодо обов'язковості прийняття рішення може виконуватися не у ста відсотках випадків. Один із способів обійти цю перешкоду – використовувати паузи (тайм-аути). Якщо не прийнято рішення про спільне значення в системі, можна взяти тайм-аут і після його закінчення почати процес досягнення консенсусу спочатку. Цей принцип використовують такі алгоритми, як Paxos і Raft. Інший підхід – зняти вимогу того, що консенсус має бути детермінованим. Цей підхід використовується в алгоритмі Бен-Ора [5].

Постановка завдання. Розробити алгоритм роботи брокеру повідомлень, який гарантує порядок доставки повідомлень, що включає в себе також гарантії відмовостійкості та захист від втрати даних.

Виклад основного матеріалу. Алгоритми консенсусу використовуються в брокерах повідомлень для обрання лідера або узгодження значення реплікаційного фактору, що має безпосередній вплив на гарантування порядку доставлення повідомлень.

Визначимо систему, яка охоплює видавця / -ів, а також одержувача / -ів повідомлень. В простому випадку одного видавця та одного одержувача система виглядала б так: один видавець передає повідомлення до черги повідомлень, на яку підписаний один одержувач. Визначення порядку повідомлень ускладнюється зі збільшенням кількості видавців або одержувачів. У ситуації, коли кілька одержувачів підписані на певну чергу повідомлень (або тему), навіть якщо повідомлення отримуються з теми в правильному порядку, немає гарантій, що цей порядок буде збережений, коли повідомлення будуть оброблятися одержувачами. Якщо порядок обробки важливий, то одержувачам потрібно буде координуватись через деяку систему зберігання ACID.

Аналогічно, кілька видавців, що відправляють повідомлення на одну і ту ж тему, унеможливлюють збереження порядку повідомлень.

Як визначити порядок повідомлень, що опубліковані від різних видавців? Або самі видавці повинні скоординуватися, або сама служба доставки повідомлень повинна приєднувати порядок замовлення до кожного вхідного повідомлення. Кожне повідомлення повинно містити інформацію, що визначає його місце в загальному порядку повідомлень. Це може бути або часовою позначкою (яку отримують усі сервери з одного джерела, щоб уникнути проблем розсинхронізації часу), або порядковим номером (отриманий з одного джерела з гарантіями ACID). Проте порядок отримання відповіді від вузла, який вираховує час, може не співпадати з порядком відправлення до нього запитів, а координація незалежних годинників на вузлах системи стикається з проблемою «дрейфу годин» – явища, за якого годинник відраховує час з трохи різною швидкістю [4].

Тому для вирішення проблеми гарантованої доставки повідомлень в системі «видавець / споживач» пропонується використовувати одного видавця, що надсилає повідомлення через один сервер (в одну тему) до одного одержувача. Асинхронна обробка зберігається в рамках поділу теми на підрозділи, де повідомлення групуються за певним критерієм та обробляються паралельно по групах. Порядок повідомлень зберігається в рамках підрозділу. Для створення реплік та гарантій збереження даних за відмови видавця, одержувача або вузла, де формується черга повідомлень, пропонується використати алгоритм консенсусу Raft для визначення лідера-видавника та лідера-споживача.

Raft – алгоритм, який працює з поправкою на те, що консенсус не завжди може бути досягнуто у фіксований час. Важливою особливістю, що відрізняє Raft, є використання загального (колективного) тайм-ауту для отримання результату (прийняття рішення). У Raft, якщо стався збій і перезавантаження, необхідний період очікування як мінімум в один раунд тайм-ауту перед тим, як буде зроблена нова спроба оголосити лідера, і таким чином гарантовано успішне отримання рішення.

Результат застосування Raft еквівалентний результату застосування Paxos[1]. Raft є таким же ефективним, як і Paxos, але його структура відрізняється. Raft відокремлює ключові елементи консенсусу, такі як вибори керівника, реплікацію журналу та забезпечення безперебійності, і забезпечує більш високий ступінь узгодженості для зменшення кількості станів системи, які необхідно враховувати. Raft також включає новий механізм зміни членства в кластері, який використовує перекриття кворумів для гарантування безпеки.

За допомогою Raft всі читання та записи проходять через лідера, завдання якого – здійснити

реплікацію записів на вузли послідовників. Коли клієнт намагається зробити читання / запис з вузла послідовника, йому повідомляється, хто є лідером, і що він має надсилати всі записи до цього вузла. Лідер підтвердить клієнту те, що запис було здійснено, лише після того, як кворум вузлів-послідовників підтвердив, що вони записали дані на сервер. Кворум – це більшість вузлів. Кластер з трьох вузлів має кворум з двох, кластер з п'яти має кворум з трьох. Кожен вузол зберігає всі повідомлення в структурі журналу, де кожне повідомлення має свій індекс. Завдання алгоритму консенсусу – забезпечити, щоб усі вузли мали однакові повідомлення з однаковими індексами у своєму журналі [6].

Raft включає в себе такі основні дії:

- вибори лідера;
- реплікація журналу.

Вибори лідера — це процес узгодження лідера. Raft не допускає одночасно двох функціональних лідерів.

Кожен вузол може бути в одному з трьох станів:

Послідовник. У цьому стані вузол не видає запитів, але пасивно чекає надходження запитів від лідера чи кандидатів.

Кандидат. Використовується для виборів лідера. Коли послідовник виявить, що зв'язок з лідером втрачено, він переходить у стан кандидата та починає надсилати запити на голосування на всі інші вузли.

Лідер. Відповідальний за взаємодію з клієнтами та реплікацію журналу до послідовників.

Коли вузол запускається, він набуває статусу послідовника і чекає серцебиття від лідера. Якщо час очікування вийшов, він набуває статусу кандидата і надсилає запит на голосування всім членам кластеру. Якщо інші вузли також вичерпали час очікування серцебиття лідера, вони можуть одночасно надсилати запити на голосування. Якщо кандидат не отримає більшість голосів, то вузол переходить у стан «Послідовник». Тільки якщо вузол отримає більшість голосів, він стане лідером. Перший вузол, який надсилає свої запити на голосування, зазвичай стає лідером.

Як тільки вузол стає лідером, він посилає періодичне серцебиття всім послідовникам. Якщо лідер виходить з ладу або зупиняється, то коли в послідовників сплине період очікування отримання серцебиття від лідера, вони знову почнуть відправляти запити на голосування, і один з них стане новим лідером.

Ера – монотонний лічильник, який використовується для виявлення застарілих вузлів, які, можливо, були вимкнені деякий час і мають застарілу інформацію. Кожен раз, коли послідовник стає кандидатом, він збільшує значення своєї ери і включає її у свої запити на голосування. Після обрання лідера ера не змінюється до наступних виборів. Поточна ера включена у всі повідомлення, якими обмінюються вузли, як механізм безпеки.

Для перемоги на виборах є три правила:

1. Більшість вузлів повинні відповісти позитивним голосом.

2. Коли кандидат надсилає запит на голосування, він включає до нього значення терміну, який він вважає поточним, + 1. Приймальний вузол перевіряє, що термін більший за його власний. Це запобігає здобуттю лідерських позицій ушкодженими вузлами (які, можливо, були відключені протягом деякого часу).

3. Кандидат включає в запит на голосування останній індекс свого журналу. Вузол буде відхиляти повідомлення, якщо індекс менший за його власний індекс останнього запису до журналу. Це гарантує, що вузол, який не має останніх записів, не може стати лідером, оскільки це може призвести до втрати даних.

Реплікація журналу виконується лідером, який надсилає своїм послідовникам повідомлення на додання записів. Послідовники вказують лідеру індекс свого останнього запису, і лідер надсилає повідомлення на додання записів, починаючи з указаного індексу. Це означає, що лідер може здійснювати реплікацію записів журналу як до послідовника, який містить майже всі останні записи, так і до послідовника, який щойно приєднався і не має даних взагалі. Повідомлення на додання записів виконує функцію серцебиття і періодично надсилається всім послідовникам, навіть якщо нових даних немає. Таким чином відбувається запис повідомлень від лідера-видавника до журналів всіх послідовників, а після отримання підтвердження від кворуму і до відповідної кількості реплік теми в черзі повідомлень. Аналогічно відбувається зчитування даних споживачами. Лідер-споживач передає послідовникам інформацію про останні опрацьовані повідомлення для того, щоб в разі його зупинки новий лідер мав інформацію про індекс останньої обробленої партії повідомлень та знав, з якого моменту продовжувати зчитування з черги повідомлень.

Однією з вимог до Raft є те, що коректність роботи алгоритму не повинна залежати від часу: система не повинна давати невідповідних результатів лише тому, що якась подія відбувається

швидше або повільніше, ніж очікувалося. Однак доступність (здатність системи своєчасно реагувати на клієнтів) неминуче повинна залежати від часу. Наприклад, якщо обмін повідомленнями триватиме довше, ніж типовий час між аваріями сервера, часу функціювання кандидатів не вистачить для виграшу виборів, а без стійкого лідера Raft не може працювати.

Вибори лідерів – це аспект алгоритму, де дотримання термінів є дуже важливим. Raft зможе обирати та підтримувати стабільного лідера в тому випадку, якщо система буде виконувати наступні вимоги щодо термінів:

Час передачі «тайм-аут виборів» MTBF

У цій нерівності час передачі – середній час, необхідний для того, щоб сервер передавав повідомлення паралельно кожному серверу в кластері та отримав від них відповіді; тайм-аут вибору – час, необхідний для голосування та вибору лідера; MTBF – це середній наробіток між відмовами для одного сервера. Час передачі повинен бути на порядок меншим за тайм-аут виборів, щоб лідери могли надійно надсилати серцебиття та запобігати ініціації повторних виборів; зважаючи на те, що тайм-аут виборів для кожного вузла встановлюється випадково з певного фіксованого інтервалу, ця нерівність робить поділ голосів маловірогідним. Тайм-аут виборів має бути на кілька порядків меншим, ніж MTBF, щоб система стабільно працювала.

В ситуації, коли необхідно реорганізувати мережу – змінити конфігурацію, додавши або відокремивши певні вузли, вузли потенційно можуть перейти в режим split-brain. Щоб механізм зміни конфігурації був безпечним, під час переходу не повинно бути жодного моменту, коли можливо обирати двох лідерів з однаковою епохою. Будь-який підхід, коли сервери переходять безпосередньо від старої конфігурації до нової, є небезпечним. Неможливо атомарно перемикнути всі сервери відразу, тому кластер потенційно може розколотися на дві незалежні частини з кворумами під час переходу.

З метою забезпечення безпеки зміна конфігурації повинна відбуватися в дві фази [1].

Спочатку кластер переходить до перехідної конфігурації, яка називається консенсус стику. А після того, як консенсус стику був записаний кворумом, система переходить до нової конфігурації.

Консенсус стику поєднує в собі і стару, і нову конфігурацію:

1) записи журналу реплікуються на всі сервери в обох конфігураціях;

2) будь-який сервер з будь-якої конфігурації може стати лідером;

3) досягнення згоди (для виборів та запису) вимагає окремих кворумів від вузлів зі старої та нової конфігурацій. Консенсус стику дозволяє окремим серверам переходити між конфігураціями в різний час, що не загрожує безпеці. Крім того, консенсус стику дозволяє кластеру продовжувати обслуговувати клієнтські запити протягом усієї зміни конфігурації.

Конфігурації кластерів зберігаються та передаються за допомогою спеціальних записів у журналі.

Коли лідер отримує повідомлення про зміну конфігурації, він зберігає та передає послідовникам конфігурацію консенсусу стику – C<old,new>. Сервер завжди використовує найновішу конфігурацію у своєму журналі, щоб приймати рішення, навіть якщо запис кворумом послідовників ще не здійснено. Коли консенсус стику записано кворумом, лідерами можуть стати лише сервери з C<old, new> у своїх журналах.

Тепер для лідера безпечно створити запис у журналі, що описує C<new>, і передати його послідовникам. Знову ж таки, ця конфігурація набуде чинності на кожному сервері, як тільки вона з'явиться. Коли нова конфігурація була записана кворумом послідовників за правилами C<new>, стара конфігурація вже не має значення, і сервери, які за новою конфігурацією мають бути відключені, можуть бути відключені.

Проте постає проблема видалених серверів (ті, які не входять до C<new>), які можуть порушити роботу кластеру. Ці сервери не отримуватимуть серцебиття лідера, тому після закінчення часу очікування вони почнуть надсилати запит на вибори з новою епохою, і це призведе до того, що поточний лідер перейде до стану послідовника. Зрештою, нового лідера буде обрано, але видалені сервери по тайм-ауту знову почнуть розсилати запит на вибори та процес повториться, що призведе до поганої доступності брокера.

Щоб запобігти цій проблемі, сервери ігнорують запит на вибори, поки вони вважають, що лідер функціонує. Зокрема, якщо сервер отримує запит на вибори в межах мінімального часу очікування виборчих слухань після отримання серцебиття від поточного лідера, він не оновлює свою епоху і не надає свій голос.

Це не впливає на звичайні вибори, де кожен сервер чекає принаймні мінімальний час очікування виборчих слухань перед початком виборів. Однак це допомагає уникнути збоїв на видалених серверах: якщо лідер може надіслати серцебиття

до кластеру, тоді він не буде переобраний за рахунок запитів на вибори з більшим значенням ери.

Висновки. Задача гарантування порядку доставки повідомлень в хмарних системах напряму пов'язана з розв'язанням проблеми досягнення консенсусу та загального порядку передачі в розподілених системах. На практиці ця проблема вирішується через пом'якшення вимог до досягнення консенсусу. Є два підходи: допускається або отримання недетермінованого консенсусу, або недосягнення консенсусу у певних випадках. Для гарантування порядку доставки для брокерів повідомлень пропонується використати схему «один видавець, що надсилає повідомлення через один сервер (до однієї теми) до одного одержувача», поєднану з алгоритмом консенсусу Raft, який використовує тайм-аути для боротьби з випадками недосягнення консенсусу.

Список літератури:

1. In search of an understandable consensus algorithm (extended version) / Diego Ongaro, John K. Ousterhout. *USENIX Annual Thechnical Conference* – 2014.
2. Impossibility of Distributed Consensus with One Faulty Process / Michael Fisher, Nancy Lynch, Michael Paterson. *Journal of the Assccktion for Computing Machinery.* April 1985. Vol. 32, No. 2. pp. 374–382.
3. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. Санкт-Петербург : Питер, 2018. 640 с. : ил. (Серия «Бестселлеры O'Reilly»).
4. Ordering messages : довідкова документація. 2019. URL: https://cloud.google.com/pubsub/docs/ordering.
5. Preethi Kasireddy How Does Distributed Consensus Work? : стаття. 2018. URL: https://medium.com/s/story/lets-take-a-crack-at-understanding-distributed-consensus-dad23d0dc95.
6. Vanlightly J. Quorum Queues Internals – A Deep Dive [Електронний ресурс]: стаття – 2019. URL: https://www.cloudamqp.com/blog/2019-04-03-quorum-queues-internals-a-deep-dive.html.
7. Yovtchev Tsviatko SQS FIFO Queues: Message Ordering and Exactly-Once Processing Guaranteed? : стаття. 2019. URL: https://www.ably.io/blog/sqs-fifo-queues-message-ordering-and-exactly-once-processing-guaranteed/.

Shaidurova K.A. GUARANTEED MESSAGE DELIVERY ORDER IN THE CLOUD-BASED SYSTEMS

One of the most important abstractions of cloud-shared systems is consensus – consistency between all nodes regarding a particular issue. Consensus is a key factor that is needed to provide order guarantees during message delivery. Consensus is also used for a variety of purposes – in the replication process, for fault-tolerance guarantees, and for data safety. In cloud-based distributed systems you need to take into account the possibility of a node failure and add a build-in mechanism that provides fault-tolerance and order guarantees.

The ordering guarantees during message delivery is important in cases where the availability is less crucial than the right ordering, for example, in bank account operations or bidding operations at auctions. This article contains the analysis of different consensus algorithms and their requirements. It also discloses the work of a message broker that guarantees message ordering with a help of Raft consensus algorithm. The article contains the description of leader-election mechanism among nodes of publishers and consumers in the clusters to ensure a correct replication and fault-tolerance with a help of reelection process in case of a leader node failure. In addition, the article shows a process of cluster change that uses a commit of an intermediate configuration before a new configuration is written to the nodes, which stops two leaders from functioning at the same time. In the article the preventative mechanism is described that does not allow new election votes for a small time period after the leader heartbeat has been received, which prevents nodes, that were deleted from the cluster, from constantly starting reelection process.

Key words: *consensus, total order transaction, fault tolerance, message broker, replication, Raft.*