УДК 004.75

А.В. Миргородський[1]
О.В. Романюк[1]

РОЗРОБКА РОЗПОДІЛЕНИХ СИСТЕМ З ВИКОРИСТАННЯМ АЛГОРИТМУ КОНСЕНСУСУ RAFT

[1]Вінницький національний технічний університет

Анотація

Наведено короткі відомості про алгоритм консенсусу Raft, його призначення та принципи роботи. Розглянуто потенційні недоліки й відкриті питання при розробці розподілених систем.

Ключові слова: ІТ, розподілені системи, відмовостійкість, висока доступність, алгоритм консенсусу, Raft.

Abstract

Brief information about the Raft consensus algorithm, its purpose and principles of operation are presented. The potential drawbacks and open issues in the development of distributed systems are considered.

Keywords: IT, distributed systems, fault tolerance, high availability, consensus algorithm, Raft.

Вступ

Сучасні обчислювальні системи дедалі частіше розробляють з орієнтацією на високу доступність, відмовостійкість та використання горизонтального масштабування. Застосування цих концепцій дозволяє програмній системі проводити менше часу в непрацездатному стані, а бізнесу та промисловості – втрачати значно менше фінансів через такі неполадки. Головна особливість архітектури з високою доступністю – це використання додаткового (надлишкового) апаратного забезпечення, яке дублює ті чи інші функції системи й зможе з мінімальною затримкою замінити основний компонент у випадку аварійної ситуації чи його апаратної або програмної відмови [1].

Основною проблемою таких розподілених або децентралізованих систем є синхронізація даних та забезпечення єдиного стану, щоб у випадку аварійної ситуації вцілілі сервери з кластеру могли продовжувати виконувати свої задачі та не виникало проблем з цілісністю даних [2, 3]. Raft є одним з популярних алгоритмів консенсусу, який вирішує питання забезпечення згоди між учасниками системи.

Метою дослідження є вивчення переваг та особливостей алгоритму консенсусу Raft при розробці розподілених систем.

Об'єктом дослідження є процес розробки розподілених систем з використанням алгоритму Raft.

Предмет дослідження – алгоритм консенсусу Raft.

Основна частина

Синхронізація даних та підтримка кластера машин в узгодженому стані може стати нетривіальною проблемою при асинхронній роботі вузлів системи або при нестабільному мережевому з'єднанні. Наприклад, для мінімізації часу непрацездатності додатку при проблемах з базою даних логічним рішенням буде створити декілька додаткових серверів-реплік, які зможуть замінити основну базу даних (БД) у критичній ситуації. В звичайному режимі додаток працює лише з однією базою, яка також регулярно надсилає нові дані у асинхронному режимі (задля кращої швидкодії) іншим базам-реплікам.

Перша проблема полягає в тому, що без використання алгоритмів консенсусу або певного механізму примусової синхронізації послідовність даних може втратити свою упорядкованість та цілісність. На основній БД транзакції будуть у правильному порядку, а на резервних – можуть мати місце перестановки або ж повна втрата окремих записів. Таким чином, коли основний сервер не доступний, то визначення нового потенційного лідера кластера може ускладнитись – сервери-репліки будуть мати різні версії стану системи.

Якщо першу проблему можна частково вирішити за допомогою перевірки часу виконання

транзакції та інших механізмів, то друга – нестабільне мережеве з'єднання між серверами, може призвести до ситуації, коли жодна з машин кластеру не буде мати повний набір даних. В такому випадку також необхідно використати певний алгоритм для визначення нового лідера кластера та відновлення фрагментованих даних.

Алгоритм консенсусу Raft призначений для вирішення цих та інших проблем розподілених систем. Він спроектований у якості заміни Paxos – більш старого сімейства алгоритмів консенсусу, які широко використовувались раніше, але були складні в розумінні та реалізації. Різноманітні реалізації Raft використовуються в таких популярних продуктах як MongoDB, Etcd, Neo4j та інших.

В основі Raft лежить скінченний автомат та набір механізмів для вибору поточного лідера кластера та реплікації даних. Кожен сервер в кластері може приймати один з трьох станів [4]:

• лідер – це єдиний вузол кластера, що може приймати й обробляти запити клієнтів; лідер може змінювати загальний стан системи та активно взаємодіє з іншими вузлами;

• послідовник – коли кластер має активного лідера, то більшість інших вузлів будуть виконувати роль послідовника; це пасивні вузли, які самостійно не ініціюють жодних запитів до інших машин кластера, вони лише приймають інструкції від лідера;

• кандидат – якщо через певний інтервал часу послідовник втрачає зв'язок з лідером, то переходить в активний стан і починає розсилати іншим вузлам запити на голосування за нового лідера.

Можливі варіанти зміни станів наведено на рисунку 1.

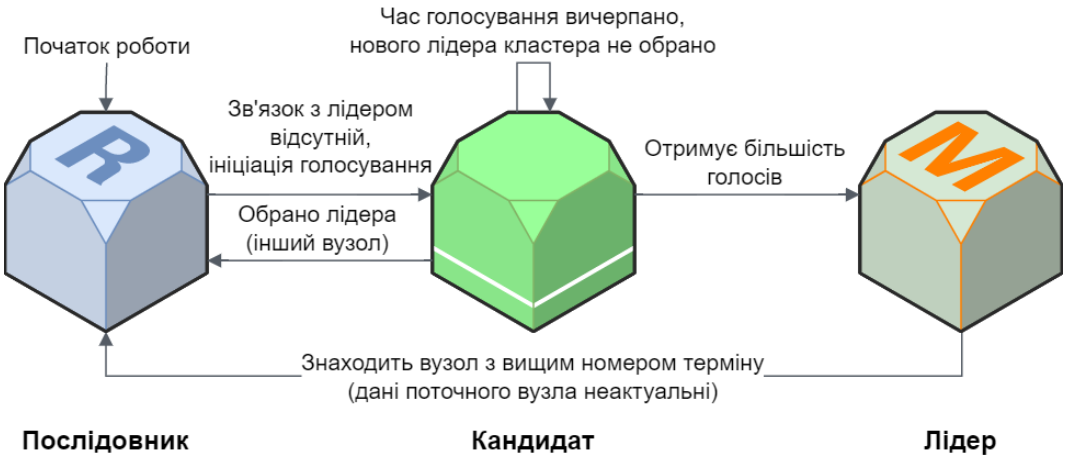

Рисунок 1 – Зміна станів в алгоритмі консенсусу Raft

Використання лише скінченного автомата не достатньо для досягнення консенсусу, адже кандидатом на нового лідера кластера може виступати вузол із застарілими даними, які не відображають реальний стан системи та можуть призвести до некоректної обробки запитів клієнта. Для вирішення цієї проблеми Raft використовує поняття термін або артикул (англ. term) – це ідентифікатор у вигляді монотонно зростаючого значення, який застосовується для глобального впорядкування подій [4]. Термін кожного вузла оновлюється лише у випадку початку виборів нового лідера або ж в результаті комунікації з іншими вузлами, коли сервер отримує більш актуальну інформацію від інших машин кластера.

Кожен вузол-послідовник має певний визначений період часу, протягом якого він очікує на один з двох запитів: інструкції від поточного лідера або ж запит на голосування від нового кандидата [5]. Якщо протягом цього періоду послідовник не отримає жодного звернення, то змінить свій стан на кандидата й самостійно почне розсилати запити на голосування. Отримання вузлом-кандидатом більшості голосів дозволить йому стати новим лідером кластера. При значних мережевих проблемах або виходу з ладу апаратного забезпечення може вийти ситуація, коли відразу неможливо досягти більшості – тоді вузли-кандидати розпочнуть нове голосування. Після закінчення виборів решта кандидатів переходять в стан послідовника.

Наступний важливий механізм роботи алгоритму Raft – це реплікація даних. Кожен вузол системи має журнал, що містить записи про всі команди, які виконує лідер при взаємодії з клієнтами. Саме цей журнал і є описом стану кластера, який підлягає дублюванню. Процес модифікації стану складається з двох частин: самої реплікації даних та закріплення нового запису. Лідер кластера при обробці запиту

користувача додає новий допис в журнал і намагається надіслати вузлам-послідовникам відповідний запит, щоб вони також додали нові дані – це реплікація. Якщо більшість вузлів змогли це зробити й відправили лідеру підтвердження, то починається закріплення допису– лідер у себе фіксує новий стан журналу й надсилає відповідні запити послідовникам [5]. На цьому реплікація даних закінчується й у випадку виходу з ладу лідера інші вузли зможуть продовжити роботу з найбільш актуальними даними. Якщо через мережеві або інші проблеми певні вузли пропустили реплікацію даних, то при синхронізації з лідером вони можуть порівняти вміст журналів і надіслати відповідне повідомлення лідеру. Тоді процес реплікації повториться для цих окремих серверів.

Не дивлячись на те, що Raft розроблявся в якості простої та ефективної заміни більш застарілим алгоритмам консенсусу, він не позбавлений власних недоліків та обмежень. По-перше, клієнти повинні працювати лише з поточним лідером кластера, тому необхідний додатковий механізм його визначення. Також така архітектура може стати значним обмеженням для високонавантажених систем – велика кількість трафіку, яка повинна проходити через одну машину-лідера, підвищує апаратні вимоги до серверів [6]. По-друге, алгоритм вирішує лише проблеми консенсусу між вузлами, а питання фактичного розподілення навантаження і задач між серверами залишається відкритим для кінцевих розробників. Крім того, не вирішеними є задачі захисту кластера від порушників (наприклад, можлива атака за допомогою додавання в кластер стороннього вузла з метою злому й несанкціонованого доступу до стану системи), а також задача функціонування системи лише з одним вузлом, що часто використовується в розподіленому програмному забезпеченні.

Висновок

Дослідження показало, що алгоритм консенсусу Raft складається з декількох простих концепцій, які легко реалізувати в реальних програмних системах – це скінченний автомат з лише трьома станами та кілька механізмів для визначення лідеру кластера та реплікації даних. Алгоритм дозволяє будувати відмовостійкі розподілені системи, але залишає відкритими питання щодо розподілення навантаження, захисту кластера та деяких інших прикладних задач.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Jevtic G. What Is High Availability? [Електронний ресурс] / Goran Jevtic // PhoenixNAP. – 2022. – Режим доступу до ресурсу: https://phoenixnap.com/blog/what-is-high-availability.
2. Firesmith D. System Resilience: What Exactly is it? [Електронний ресурс] / Donald Firesmith // Carnegie Mellon University, Software Engineering Institute. – 2019. – Режим доступу до ресурсу: https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/.
3. Пєтух А.М.,Романюк О.Н., Романюк О.В. «Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних», Електронний навчальний посібник. ВНТУ, 2016. – Режим доступу: http://ir.lib.vntu.edu.ua//handle/123456789/34313
4. Howard H. ARC: Analysis of Raft Consensus [Електронний ресурс] / Heidi Howard // University of Cambridge Computer Laboratory. – 2014. – Режим доступу до ресурсу: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-857.pdf.
5. Raft Refloated: Do We Have Consensus? [Електронний ресурс] / H.Howard, M. Schwarzkopf, A. Madhavapeddy, J. Crowcroft // University of Cambridge Computer Laboratory. – 2015. – Режим доступу до ресурсу: https://www.cl.cam.ac.uk/~ms705/pub/papers/2015-osr-raft.pdf.
6. Hooda P. Raft Consensus Algorithm [Електронний ресурс] / Parikshit Hooda // GeeksforGeeks. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/raft-consensus-algorithm/.

Миргородський Андрій Вікторович – студент групи 3ПІ-22м, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: mirgorodskijav@gmail.com

Романюк Оксана Володимирівна – к.т.н., доцент кафедри програмного забезпечення, Вінницький національний технічний університет, м. Вінниця, e-mail: romaniukoksanav@gmail.com

***Myrhorodskyi Andrii* –** student of group 3PI-22m, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: mirgorodskijav@gmail.com

Oksana Romaniuk – Candidate of Technical Sciences, Associate Professor of the Software Chair, Vinnytsia National Technical University, Vinnytsia, e-mail: romaniukoksanav@gmail.com