УДК 621.396.004 **DOI:10.30837/rt.2024.2.217.10**

Л.О. ТОКАР, канд. техн. наук, В.Є. ЦИЛЮРИК, В.В. СОЛОДІЛОВ

ДОСЛІДЖЕННЯ ПРОЦЕСУ РЕПЛІКАЦІЇ ДАНИХ ЗА ДОПОМОГОЮ АЛГОРИТМА РЕПЛІКАЦІЇ RAFT ДЛЯ ПІДТРИМКИ УЗГОДЖЕНОСТІ В КЛАСТЕРІ СЕРВЕРІВ

Вступ

Віртуальні IP-АТС є популярним рішенням для організацій, які хочуть мати гнучкість та контроль своїх комунікаційних сервісів. Для забезпечення максимально ефективного та надійного використання технологій віртуальних IP-АТС у складних розподілених середовищах, а також для підвищення якості послуг та зниження загроз зупинки роботи системи у випадках збоїв або атак, використовуються методи та механізми, які є вирішальними для забезпечення стійкості комунікаційних сервісів.

Методи та технології забезпечення високої доступності та відмовостійкості для віртуальних сервісів є важливими компонентами сучасних інформаційних структур. Завдяки їм компанії можуть забезпечувати безперебійну роботу своїх систем і служб, незалежно від можливих відмов та збоїв [1]. Вирішення питань забезпечення стійкої роботи віртуальних IP-АТС полягають у кластерізації серверів, використанні методів контейнеризації, механізмів управління та контролю та механізмів реплікації для забезпечення консистентності даних у кластері.

Для забезпечення відмовостійкості використовуються резервне копіювання та реплікація даних. Це дозволяє відновлювати систему після відмови, не втрачаючи важливої інформації. Висока доступність забезпечується за допомогою реплікації даних, автоматичного перемикання між серверами або дата-центрами та швидкого відновлення після відмов. Реплікація не тільки підвищує надійність системи, але й покращує продуктивність, оскільки запити до даних можуть оброблятися локально на кожному вузлі, зменшуючи час відгуку та навантаження на мережу.

Враховуючи важливість реплікації даних, особливу увагу слід приділити безпеці. Репліковані дані повинні бути захищені від несанкціонованого доступу та змін, особливо у випадках, коли вузли кластера розташовані в різних фізичних місцях або використовуються хмарні послуги. Це вимагає впровадження сильних механізмів автентифікації, шифрування та контролю доступу. З огляду на сучасні вимоги до масштабованості та доступності великих розподілених систем, реплікація є ключовим компонентом, що забезпечує ефективну та надійну роботу додатків та послуг. Це й обумовлює актуальність даної роботи.

Кластеризація, яка дозволяє групувати кілька серверів у єдиний віртуальний ресурс, є одним з ключових методів досягнення високої доступності. Кластеризація на основі реплікації є ключовим елементом у сучасних розподілених системах, забезпечуючи високий рівень доступності та відмовостійкості. Основна ідея полягає у створенні декількох копій даних на різних вузлах кластера, що дозволяє системі продовжувати працювати без перебоїв навіть у випадку збоїв або відмов окремих компонентів [2].

Окрім вибору стратегії реплікації, важливим аспектом є управління навантаженням та моніторинг стану різних вузлів у кластері. Ефективне розподілення навантаження між вузлами може значно покращити загальну продуктивність та доступність системи. Це включає не тільки рівномірний розподіл запитів між вузлами, але й адаптацію до змін у навантаженні та можливих відмовах компонентів. Моніторинг стану вузлів дозволяє оперативно виявляти та реагувати на проблеми, забезпечуючи безперервну роботу системи.

Сучасні рішення для забезпечення високої доступності та відмовостійкості також включають в себе георедунтантність, коли дата-центри розташовані в різних географічних локаціях. Це не тільки розподіляє ризики, але і дозволяє забезпечити безперебійний доступ

до ресурсів для користувачів з різних регіонів. Прикладом є розгортання дата-центрів в Amazon Web Services (AWS).

Важливо зазначити роль програмного забезпечення у забезпеченні високої доступності та відмовостійкості. Сучасні системи управління даними, такі як бази даних та системи управління контентом, часто включають в себе механізми реплікації, кешування та резервного копіювання, що допомагає збільшити стабільність та доступність даних [3].

Забезпечення високої доступності та відмовостійкості потребує комплексного підходу, який поєднує апаратні, програмні та організаційні рішення. Важливими моментами є регулярність в оновлювленні технологій, проведення аудиту систем на предмет потенційних слабких місць й навчання персоналу найкращим практикам у галузі забезпечення безперебійної роботи віртуальних інфраструктур.

Основна частина

Реплікація даних є важливою стратегією для забезпечення надійності, доступності та продуктивності систем обробки даних. Цей процес полягає у створенні та підтримці копій даних на різних серверах або вузлах, щоб забезпечити можливість доступу до даних у випадку відмови обладнання та програмного забезпечення чи інших непередбачуваних подій.

Реплікація може бути використана для забезпечення доступу до даних у випадку відмови сервера, підвищення продуктивності через розподілення навантаження або для забезпечення географічної резервності даних.

Відомі різні типи реплікації: повна реплікація, часткова реплікація, синхронна реплікація, асинхронна реплікація, які відрізняються способами копіювання та різним ступенем доступності даних.

Високий рівень відмовостійкості і доступності даних забезпечує метод реплікації на рівні блоків, що дозволяє створити точні копії даних на різних пристроях. У цьому методі зміни, що відбуваються у блоку даних на одному диску, автоматично реплікуються на інший диск. У хмарних сервісах дані також можуть реплікуватися на різні сервери [4].

Даний метод широко використовується у системах Redundant Array of Independent Disks (RAID), які гарантують, що навіть у разі відмови одного диску дані залишаються доступними та незмінними (рис. 1).

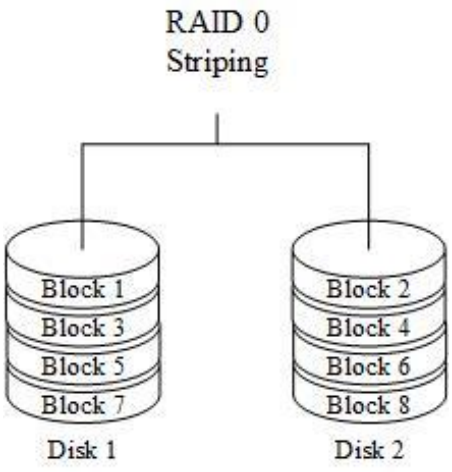

Рис. 1. Діаграма блочної реплікації у системі RAID

Такий підхід до реплікації має широкий спектр застосувань. Його може бути використано для оптимізації роботи з базами даних чи в системах з розподіленою файловою системою, де реплікація блоків даних допомагає забезпечити надійність та швидкодіючий доступ до інформації.

Об'єктна реплікація – це метод реплікації даних, який копіює або синхронізує об'єкти даних (які можуть включати файли, блоки даних чи інші структуровані формати) між різними системами зберігання (рис. 2).

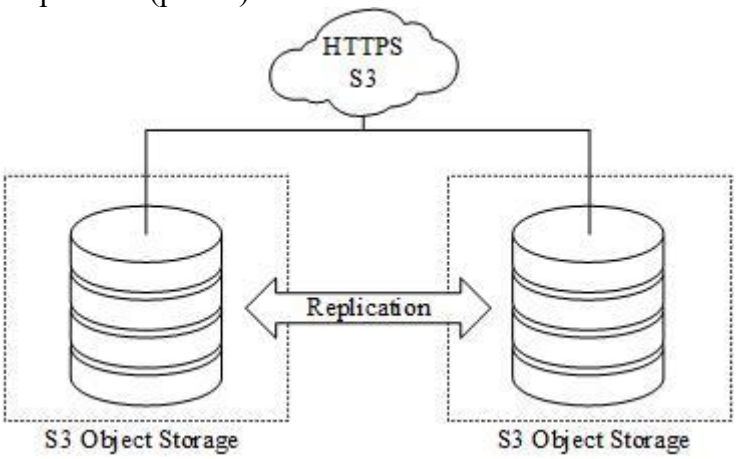

Рис. 2. Діаграма об'єктної реплікації

Цей метод реплікації широко використовується в хмарних та розподілених системах зберігання, де необхідно забезпечити надійність та доступність великих обсягів даних. Він дозволяє легко масштабувати системи зберігання й забезпечує швидкий доступ до даних, але також може вимагати складнішого управління та забезпечення додаткового простору для зберігання.

Кожний з методів реплікації має свої переваги та обмеження, тому є певний компроміс між стійкістю отриманих даних та проблемами обмеження в швидкості чи в масштабуванні. Цілісність даних, отриманих в процесі реплікації, може обмежити швидкість запису. Легкість в масштабуванні системи зберігання й швидкий доступ до даних може вимагати складнішого управління та забезпечення додаткового простору для зберігання. Зменшення ризику втрати даних і забезпечення безперервної роботи послуг для користувачів вносять проблеми керування конфліктами, які виникають при оновленні реплік і потребують розробки механізмів для їх вирішення. Тому важливо вибрати підхід, що найкраще відповідає специфічним вимогам та характеристикам системи.

Розподілені системи, як важлива частина сучасних обчислювальних мереж, вимагають надійних механізмів для досягнення консенсусу та управління даними. Алгоритм Raft, розроблений як альтернатива складнішим алгоритмам, таким як Paxos, забезпечує ефективне та зрозуміле рішення для реплікації журналу в розподілених системах. Цей алгоритм вирішує проблему досягнення консенсусу, забезпечуючи такі умови, що кожний вузол в системі може безпечно та надійно виконувати однакові операції у встановленому порядку.

В роботі проведено аналіз літератури з використання алгоритму консенсусу Raft. В роботі [55] запропоновано використання алгоритму консенсусу Raft як більш зрозумілого й модульного підходу в порівнянні з іншими алгоритмами консенсусу.

Ймовірність поділу розподіленої мережі з використанням простого, але точного аналітичного підходу, проаналізовано у роботі [6]. Ймовірність поділу мережі представлено як функцію розміру мережі, швидкості втрати пакетів та тривалості тайм-ауту. Для перевірки використано симулятор алгоритму Raft. За допомогою розробленої моделі запропоновано теоретично передбачити час та ймовірність поділу мережі, а також оптимізувати параметри алгоритму консенсусу Raft.

Модель для імітації системи Blockchain з використанням теорії масового обслуговування запропоновано у роботі [7]. Модель створено з використанням СМО M/M/1 у якості пулу пам'яті та СМО M/M/c у якості пулу майнінгу. Цей метод простий, але ефективний для виявлення важливих показників мережі: при визначенні кількості транзакцій на блок, часу майнінгу кожного блоку, пропускної здатності системи/транзакцій в секунду, кількості пулів

пам'яті, часу очікування в пулі пам'яті, кількості непідтверджених транзакцій у всій системі, загальної кількості транзакцій та кількості згенерованих блоків.

Модифікований консенсусний алгоритм стійкості до відмови для вирішення проблем масштабованості Istanbul Byzantine Fault Tolerant (IBFT) в приватних Blockchain-системах розглянуто в роботі [8]. Використано інструмент для тестування Blockchain Hyperledger Caliper, який дозволяє оцінити продуктивність платформ з відкритим вихідним кодом, таких як Quorum та Hyperledger Fabric. Використання Fabric протоколів Solo і Raft як служби замовлення транзакцій у версії 1.4 забезпечує надійну пропускну здатність та низьку затримку, що робить Hyperledger Fabric хорошим кандидатом для корпоративного рішення на основі Blockchain. З іншого боку, реалізація Raft та IBFT у Quorum як протоколів консенсусу, які відповідають за підтримку ланцюжка блоків, робить його більш логічним варіантом для тестування модифікацій.

В роботі [9] запропоновано механізм вибору лідерів за допомогою алгоритму Raft з урахуванням стабільності мережі. Розроблений метод знизив можливість поділу мережі за допомогою Raft. Крім того, це дало змогу запобігти подальшому зниженню продуктивності, коли мережа вузлів Blockchain нестабільна.

У роботі [10] розглянуто функціональні переваги та недоліки алгоритму Raft; характеристики безпеки, довіри між учасниками, пропускної здатності та масштабованості. Виявлено, що жоден алгоритм, окрім Raft, не продемонстрував повного домінування у всіх аспектах порівняння.

У роботі [11] розглянуто СМО M/G/1 з ланцюгом Маркова за дискретним часом Discrete-Time Markov Chain (DTMC) у системі Blockchain. За допомогою теорії масового обслуговування теоретично проаналізовано процеси генерації блоків та побудови системи Blockchain. За допомогою мови програмування Python проведено знаходження середнього часу підтвердження транзакції, кількості транзакцій у черзі у точці прибуття, кількості транзакцій у точках відправлення та інші параметри.

Крім того, розглянуто марківську модель СМО Mb/M/1 як теоретичну основу для аналізу системи Blockchain на основі доказу повноважень, що фокусуються на стохастичній поведінці транзакцій та для аналізу консенсусного протоколу повноважень. Представлено дані, які проведені на Python. Зазначено, що оптимальна пропускна здатність блоку досягається при невеликій кількості транзакцій.

У контексті розподілених систем, реплікація даних є ключовим фактором для забезпечення високої доступності та стійкості системи. Raft дозволяє системам ефективно керувати реплікацією, забезпечуючи консистентність даних між вузлами. Це особливо важливо в сценаріях, де втрата або неконсистентність даних може призвести до серйозних наслідків.

Алгоритм Raft базується на концепції лідера. Всі вузли в кластері є або послідовниками (followers), або кандидатами, поки один з них не стає лідером. Лідер керує процесом реплікації даних, відправляючи оновлення журналу іншим вузлам. Такий підхід спрощує процес реплікації та знижує ймовірність розбіжностей даних.

Розробка алгоритму Raft вмотивовано потребою в більш зрозумілому та менш складному рішенні для досягнення консенсусу в розподілених системах, ніж наявні алгоритми. З часом Raft здобув популярність у спільноті розробників завдяки своїй простоті та ефективності. Його чітка структура та легкість у розумінні зробили його популярним вибором для багатьох великих розподілених систем.

Зазвичай архітектура розподілених систем базується на принципах мікросервісів та контейнеризації. Механізмом управління та контролю є платформа Kubernetes. Kubernetes відповідає за управління цими контейнерами, гарантує їх доступність та відмовостійкість.

У контексті Kubernetes, алгоритм Raft може бути використано для керування розподіленою системою еталонів (etcd), яка є ключовою складовою для зберігання та синхронізації конфігурації кластера Kubernetes. Etcd використовує Raft для забезпечення

сильної консистентності (strong consistency) стану кластера, що є критично важливим для забезпечення надійної та передбачуваної оркестрації контейнерів [12].

Raft дозволяє кластеру etcd працювати у якості єдиного цілісного компоненту, незважаючи на фізичну розподіленість між вузлами [13]. Це досягається завдяки реплікації журналу команд між всіма вузлами, що входять до кластера etcd, забезпечуючи сильну консистентність та високу доступність. Консенсусну архітектуру etcd з використанням алгоритму Raft показано на рис. 3.

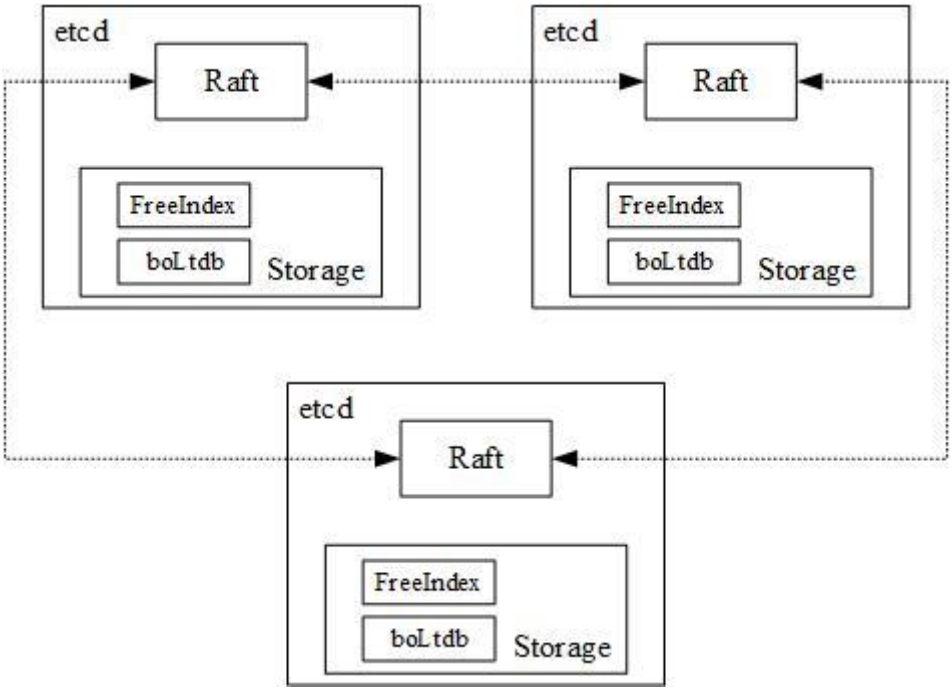

Рис. 3. Консенсусна архітектура etcd з використанням алгоритму Raft

Надійність Kubernetes в значній мірі залежить від надійності etcd, оскільки etcd зберігає важливий стан, що включає конфігурацію мережі, сервіси, політики безпеки та інше. Якщо дані в etcd не будуть консистентними або якщо сервіс стане недоступним, то це може призвести до серйозних збоїв в роботі кластера.

Raft гарантує, що навіть у разі збоїв у кількох вузлах, etcd зможе зберігати стан кластеру без втрат. Коли лідер отримує зміни стану, він не застосовує ці зміни до свого локального стану до тих пір, поки не отримає підтвердження, що зміни були репліковані на більшість вузлів. Такий підхід запобігає режиму split-brain, коли різні частини кластера можуть мати суперечливі стани.

Завдяки використанню etcd та Raft, платформа Kubernetes може забезпечити високий рівень стійкості та надійності, які необхідні для сучасних розподілених систем. Raft особливо корисний для середовищ, де потрібна сильна консистентність та здатність швидко відновлювати систему після збоїв.

Однією з основних характеристик Raft є його структура з чітко визначеними ролями (лідер, кандидат, послідовник) та простими правилами переходу між цими станами. Це робить процес лідерства та реплікації даних прозорим та передбачуваним. Крім того, використання випадкових тайм-аутів для ініціації виборів знижує ймовірність конфліктів та забезпечує ефективний вибір лідера.

На сьогодні Raft є одним з основних алгоритмів для досягнення консенсусу у розподілених системах. Його здатність забезпечувати надійну реплікацію даних та високий рівень доступності робить його незамінним інструментом в арсеналі розробників, що працюють із складними розподіленими системами.

Простота методу консенсусу алгоритму Raft робить його легшим для розуміння та застосування в порівнянні з іншими алгоритмами консенсусу. Також, Raft забезпечує кращі гарантії безпеки, ніж існуючі алгоритми консенсусу. Він гарантує, що система залишається послідовною, навіть коли відбувається кілька збоїв або розділів мережі. Відомі блокчейн-платформи, такі як R3 Corda та Quorum, що використовують Raft як протокол консенсусу.

У Raft лідер відіграє ключову роль у координації дій між різними вузлами кластера. Лідер не тільки відповідає за реплікацію журналу записів до інших вузлів, але й гарантує їх консистентність та порядок. Вибір лідера є критичним кроком у роботі Raft, оскільки від цього залежить стабільність та ефективність усієї системи.

Процес вибору лідера в Raft ініціюється, коли вузол переходить у стан кандидата. Це відбувається у випадку, якщо вузол не отримує достатньої кількості heartbeats від поточного лідера протягом певного тайм-ауту. Кандидат ініціює вибори, генеруючі голосування серед інших вузлів кластера.

Під час виборчого процесу, кандидат відправляє запити на голосування до інших вузлів у кластері. Щоб вузол став лідером, він повинен отримати більшість голосів від інших вузлів. У випадку, якщо декілька вузлів стають кандидатами одночасно, система може не досягнути консенсусу, що призведе до повторення виборчого процесу після наступного тайм-ауту. Діаграму станів вибору лідера в алгоритмі Raft показано на рис. 4.

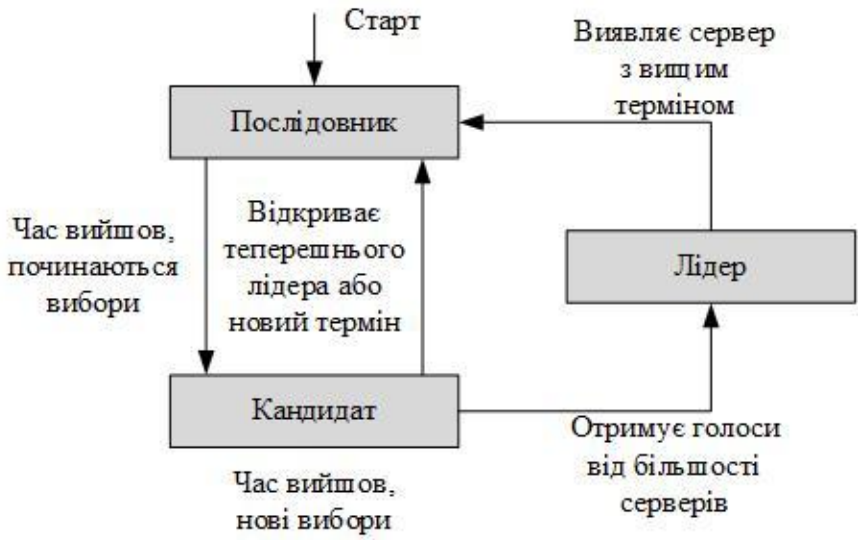

Рис. 4. Діаграма станів вибору лідера в алгоритмі Raft

Консистентність даних між вузлами в розподіленій системі в алгоритмі Raft забезпечує реплікація журналу, що є фундаментальним процесом. Цей процес відбувається під керівництвом лідера, який відповідає за зміни при веденні й синхронізацію журналу на всіх вузлах кластера. У журналі міститься серія записів, які вузли повинні включатися у визначеному порядку.

Після обрання лідера починається обслуговування запитів клієнтів. Кожен запит містить команду, яка повинна бути виконана на розподіленому консистентному автоматі. Лідер додає команду до свого журналу як новий запис, а потім паралельно для кожного сервера викликає процедуру AppendEntries, щоб реплікувати цей запис.

Лідер вирішує, коли можна виконати наступний запис у журналі розподіленого консистентного автомату. Ці записи називаються зафіксованими. Raft гарантує, що зафіксовані записи не будуть втрачено та в кінцевому підсумку будуть виконано на всіх доступних консистентних автоматах.

Raft гарантує наступні властивості: якщо два записи у різних журналах мають однаковий індекс і номер епохи, то вони містять одну й ту ж команду; якщо два записи у різних журна-

лах мають однаковий індекс і номер епохи, то всі попередні команди в цих журналах ідентичні. Невідповідності можуть накопичуватися в результаті серії відмов лідерів й вузлів.

Вузол може не мати записів, які є у лідера (так як лідер не встиг реплікувати всі записи), він може мати додаткові записи, яких немає у лідера (так як старий лідер почав реплікацію запису, але відмовив під час виконання цієї операції), а також можуть бути записи, яких немає ні в одному з серверів. У журналі вузла може бути кілька таких записів.

Таким чином, значення процедури AppendEntries досягне точки, де лідер й вузол мають ідентичні журнали. Коли це станеться, перевірка узгодженості AppendEntries успішно виконається, потім процедура видалить конфліктуючі записи в журналі вузла, а потім додасть записи з журналу лідера (якщо такі існують). Після успішного виконання процедури AppendEntries журнал вузла буде узгоджений з лідером.

В контексті роботи алгоритму Raft тайм-аути та терміни є двома фундаментальними концепціями, які сприяють стабільності та надійності реплікації даних в розподілених системах. Тайм-аути використовуються для ініціювання виборів та запобігання конфліктів, а терміни допомагають забезпечити послідовність й визначити часові рамки діяльності кластера.

Тайм-аути в Raft мають критичне значення в процесі виборів лідера. Коли вузол не отримує повідомлень від лідера протягом певного інтервалу часу (тайм-аут), він переходить у стан кандидата та ініціює нові вибори. Цей механізм гарантує, що кластер може швидко реагувати на втрату лідера та забезпечити безперервність управління даними.

Налаштування тайм-аутів є ключовим механізмом для забезпечення стабільності роботи кластера. Занадто короткі тайм-аути можуть спричинити часті та непотрібні вибори лідера, тоді як занадто довгі тайм-аути можуть затримувати відновлення системи після втрати лідера.

Процес консенсусу Raft використовується для підтримки узгодженості в кластері. Таким чином, Raft розкладає проблему консенсусу на відносно незалежні підпроблеми, які можна описати наступним чином [14]:

– вибори лідера: нового лідера потрібно обирати, коли існуючий лідер зазнає невдачі;

– реплікація журналу: лідер повинен приймати записи журналу від клієнтів і реплікувати їх по кластеру, змушуючи інші журнали погоджуватися з власним.

В контексті моделі черг процес реплікації журналу алгоритма консенсусу Raft розглядається як система черги з декількома серверами. В роботі використано модель СМО M/M/c для дослідження систем консенсусу, де прибуття моделюються як процес Пуасона, а час обслуговування має експоненційний розподіл. Модель M/M/c є однією з базових моделей у теорії масового обслуговування та використовується для аналізу систем з кількістю обслуговуючих пристроїв (серверів) більше одного. Термін M/M/c описує систему, де M відноситься до Markovian (експоненційний розподіл часів між прибуттям клієнтів та обслуговуванням), а c позначає кількість серверів у системі [15].

У процесі роботи для консенсусу Raft сервер відноситься до вузла-послідовника у розподіленій системі, який бере участь у протоколі консенсусу. Кожен сервер підтримує копію журналу системи та спілкується з іншими серверами, щоб переконатися, що вони погоджуються з поточним станом журналу. Запити клієнтів надходять до кожного вузла-послідовника згідно з процесом Пуасона з інтенсивністю λ, і вимагають часу обслуговування $1/\mu$.

Процес прибуття: як тільки лідера обрано, то він починає обслуговувати запити клієнтів. Кожен запит клієнта містить команду, яка має бути виконана реплікованими становими машинами. Лідер додає команду до свого журналу як новий запис.

Таким чином, процес прибуття моделюється як процес Пуасона з інтенсивністю, яку визначено частотою повідомлень лідера, які відправляються послідовникам. Показник λ – це інтенсивність прибуття запитів клієнта від вузла-лідера до вузлів-послідовників.

Процес обслуговування наступний: лідер додає команду до свого журналу як новий запис, потім видає Remote Procedure Call (RPC) запити AppendEntries паралельно кожному з інших послідовників для реплікації запису. Коли запис було безпечно репліковано, лідер застосовує запис до своєї станової машини і повертає результат цього виконання клієнту.

Час обслуговування також моделюється як процес Пуасона з експоненційним розподілом та з інтенсивністю, що визначено часом, який потрібний послідовникам для реплікації записів журналу лідера. Показник μ – це швидкість обслуговування кожного послідовника, що представляє швидкість, з якою записи журналу реплікуються від послідовника до лідера.

В роботі розроблено код програми для проведення експериментів на мові Python, фрагменти якого надано на рис. 5 та 6.

```
import numpy as np
import matplotlib.pyplot as plt

# Параметри, які згадані в завантаженому зображенні
mu_values = [625, 650, 675, 700]  # різні значення сервісної швидкості м
c_values = [3, 5, 11]             # різні значення кількості каналів с
lambda_range = np.arange(150, 601) # діапазон значень л

# Функція для розрахунку очікуваного часу запиту в системі (latency)
def expected_latency(lmbda, mu, c):
    if lmbda < c * mu:
        return 1 / mu
    else:
        return c / (lmbda * (c * mu - lmbda))

# Функція для розрахунку ймовірності затримки повідомлення
def probability_of_delay(lmbda, mu, c):
    return lmbda / (c * mu)
```

Рис. 5. Фрагмент коду програми на Python

```
# Графіки для різних с та м
for c in c_values:
    for mu in mu_values:
        latency = [expected_latency(lmbda, mu, c) for lmbda in lambda_range]
        axes[1, 1].plot(lambda_range, latency, label=f'c={c}, м={mu}')
axes[1, 1].set_title('Latency for different м and c')
axes[1, 1].set_xlabel('Arrival Rate л')
axes[1, 1].set_ylabel('Latency')
axes[1, 1].legend()
axes[1, 1].grid(True)
plt.tight_layout()
plt.show()
```

Рис. 6. Фрагмент коду програми на Python

Прийняття запитів клієнтів до середньої інтенсивності λ відповідно до моделі Пуассона визначено за формулою

$$\lambda_n = \lambda. \tag{1}$$

Існує c паралельних ідентичних послідовників (серверів), кожен з яких працює згідно з експоненційним розподілом з середньою швидкістю μ. Для n клієнтських запитів, швидкість обслуговування сервера може бути отримана в наступних ситуаціях:

– якщо $n \langle c$, то всі клієнтські запити можуть бути обслужено одночасно. Черга буде відсутня. Кількість простоюючих серверів у цьому випадку становить $c - n$. Тоді $\mu n = n\mu$, $n = 0, 1, 2, ..., c$;

– якщо $n \geq c$, то всі сервери зайняті та максимальна кількість клієнтських запитів, що очікують в черзі, становитиме $n - c$. Тоді $\mu_n = c_\mu$ для $n = 0, 1, 2, ...$

Таким чином, швидкість обслуговування μ_n може бути визначено:

$$\mu_n = \begin{cases} n\mu, & n \leq c \\ c\mu, & c \leq n \end{cases}. \tag{2}$$

Цей аналіз дає розуміння того, як алгоритм Raft працює в різних контекстах і може бути застосований для оптимізації процесів проєктування систем.

Продуктивність Raft може бути підвищена шляхом зміни характеристик системи, таких як, кількість вузлів-послідовників, їх обслуговуюча здатніть, надходження запитів від клієнтів тощо. Результати проаналізовано між компромісами пропускної здатності, затримки та ймовірності затримки. Це може допомогти системним дизайнерам робити практичні вибори щодо розподілу ресурсів та налаштування системи.

Ймовірність затримки повідомлення визначено:

$$P(\lambda, \mu, c) = \frac{\lambda}{c\mu}. \tag{3}$$

Очікувану затримку визначено:

$$L(\lambda, \mu, c) = \begin{cases} \frac{1}{\mu}, & \\ \frac{c}{\lambda(c\mu - \lambda)}, & \lambda \langle c\mu. \end{cases} \tag{4}$$

У роботі проведено моделювання в середовищі Visual code на Python, щоб продемонструвати зміну основних параметрів для відповідного механізму консенсусу Raft моделі технології блокчейн у контексті системи черги.

Для різних значень μ при $\lambda \in [150, 600]$ досліджено такі параметри: кількість послідовників, очікуваний час запиту клієнта у системі, тобто затримка, ймовірність затримки повідомлення тощо. На рис. 7 показано графіки обробки запитів клієнтів для $\mu = 650$.

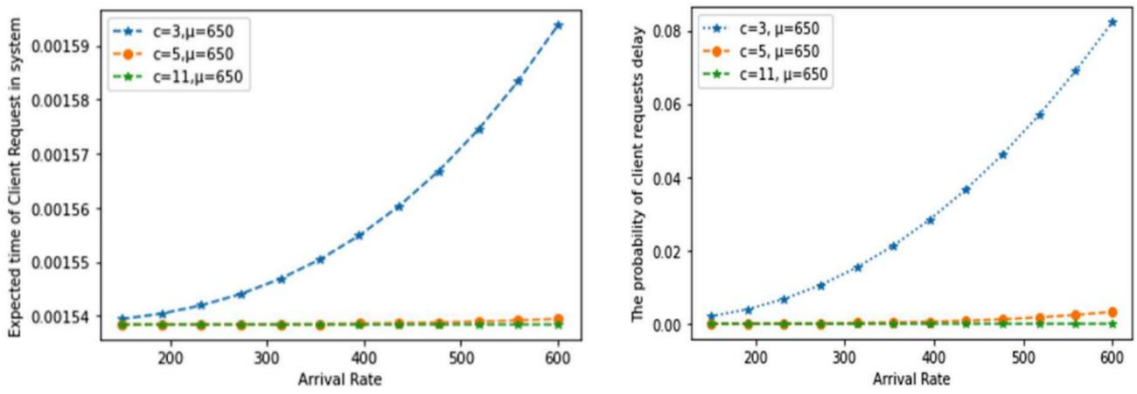

Рис. 7. Графіки обробки запитів клієнтів для $\mu = 650$

При аналізі графіків очікуваного часу запиту клієнта, який відноситься до затримки, та ймовірності затримки для $\mu = 650$, видно, що по мірі збільшення швидкості, очікуваний час запиту клієнта збільшується для $c = 3$.

Для $c = 5$ та $c = 11$ видно, що при більшій кількості послідовників параметр затримки та затримки повідомлень не залежать від швидкості прибуття.

На рис. 8 показано графіки обробки запитів клієнтів для інших значень μ.

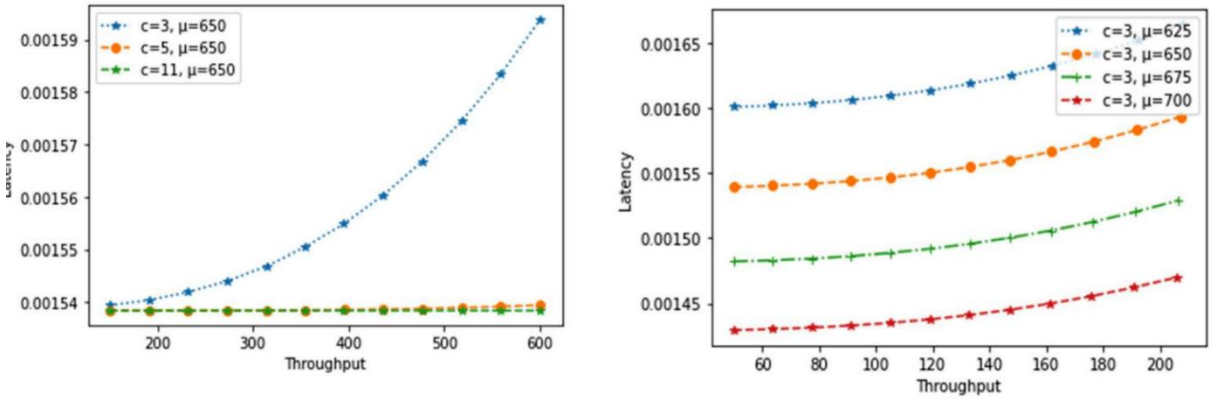

Рис. 8. Графіки очікуваного часу запиту клієнта для інших значень μ

При аналізі графіків очікуваного часу запиту клієнта, який визначається як затримка, та ймовірності затримки повідомлення для $c = 3$ та різних μ видно, що по мірі збільшення швидкості прибуття очікуваний час запиту клієнта зростає для $c = 3$ та різних μ.

На рис. 9 показано графіки аналізу очікуваного часу клієнтських запитів та ймовірності затримок у системі.

При аналізі графіків очікуваного часу запиту клієнта, який визначається як затримка, та ймовірності затримки повідомлення для комбінацій c та μ, видно, що по мірі збільшення швидкості прибуття очікуваний час запиту клієнта зростає для комбінацій c та μ.

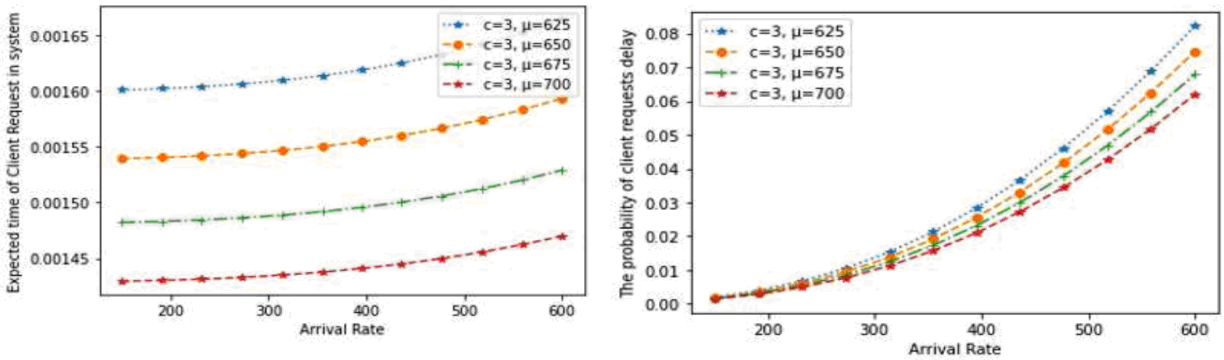

Рис. 9. Графіки аналізу очікуваного часу клієнтських запитів та ймовірності затримок у системі

Таким чином, проаналізовано показники продуктивності: очікуваний час запитів клієнтів у системі, затримка, очікувана кількість запитів клієнтів у системі та черзі, пропускна здатність, ймовірність затримки повідомлення алгоритму консенсусу Raft.

Висновки

В роботі наведено типи реплікації даних та обґрунтовано її використання для забезпечення надійності та доступності даних в розподілених системах. Проведено аналіз літератури

з використання алгоритму консенсусу Raft та наведено основи цього алгоритму. Проведено аналіз математичної моделі процессу консенсусу Raft, використано модель СМО M/M/c.

Проведено аналіз моделі M/M/c, яка є фундаментальною в теорії масового обслуговування, в якій час між прибуттям заявок та час обслуговування підпорядковуються експоненційному розподілу. Ця модель дозволяє оцінити ефективність системи кластерізації та реплікації Raft з кількома каналами обслуговування.

Проведено моделювання в середовищі Visual code на Python для дослідження зміни основних параметрів для відповідного механізму консенсусу Raft моделі технології блокчейн у контексті системи черги. Отримано графіки очікуваного часу запиту клієнта та ймовірність затримки у системі для різних значень швидкості обслуговування та кількості серверів. Доведено, що по мірі збільшення швидкості прибуття очікуваний час запиту клієнта зростає.

Список літератури:

1. DevOpsCube. How to Setup Prometheus Monitoring On Kubernetes Cluster. URL: https://devopscube.com/setupprometheus-monitoring-on-kubernetes.
2. Л.О. Токар, О.А. Колтаков, В.Є. Цилюрик. Створення тестового стенду Call-центру для балансування навантаження серверів Asterisk у кластері // Радіотехніка. 2023. № 212. С. 186–196. doi:10.30837/rt.2023.1.212.18.
3. Grafana Dashboards. Node Exporter Full. URL: https://grafana.com/grafana/dashboards/1860.
4. P. Pyda, M. Przywuski, T. Dalecki, J. Sliwa. Efficiency of Virtual Machine Replication in the Data Center // International Conference on Military Communications and Information Systems (ICMCIS 2022). 2022. Vol. 205. P. 208–217. doi:10.1016/j.procs.2022.09.022.
5. R.A. Memon, J.P. Li, J. Ahmed. Simulation Model for Blockchain Systems Using Queuing Theory // Electronics. 2019. Vol. 8 (2). P. 1–19. doi:10.1007/978-3-031-21229-1_1.
6. A. Baliga, I. Subhod, P. Kamat & S. Chatterjee. Performance evaluation of the quorum blockchain platform. URL: https://arxiv.org/abs/1809.03421.
7. Q.L. Li, J.Y. Ma, Y.X. Chang. Blockchain Queue Theory // Proceedings of the 7th International Conference on Computational Social Networks, CSoNet 2018. 18-20 Dec., 2018. Vol. 11280. P.25-40. doi:10.1007/978-3-030-04648-4_38.
8. D. Ongaro & J. Ousterhout. In search of an understandable consensus algorithm // 2014 USENIX Annual Technical Conference. 2014. P. 305–319. doi: 10.25209/2079-3316-2021-12-2-137-192.
9. D. Kim, I. Doh & K. Chae. Improved raft algorithm exploiting federated learning for private blockchain performance enhancement // 2021 IEEE International Conference on Information Networking (ICOIN). 2021. P. 828–832. doi: 10.1109/ICOIN50884.2021.9333932.
10. S. Vora, N. Thakkar, R. Gor. A Study of Performance Measures and Throughput of Raft Consensus Algorithm // International Journal for Research in pplied Science & Engineering Technology (IJRASET). 2023. Vol. 45. P. 862–869. doi:10.22214/ijraset.2023.54751.
11. G. Yang, K. Lee, Y. Yoo, H. Lee & C. Yoo. Resource Analysis of Blockchain Consensus Algorithms in Hyperledger Fabric // IEEE Access. 2022. Vol. 10. P. 74902–74920. doi: 10.1109/ACCESS.2022.3190979.
12. Q. Huo, S. Li, Y. Xie and Z. Li. Horizontal Pod Autoscaling based on Kubernetes with Fast Response and Slow Shrinkage // 2022 International Conference on Artificial Intelligence, Information Processing and Cloud Computing (AIIPCC), Kunming, China, 2022. P. 203–206. doi: 10.1109/AIIPCC57291.2022.00051.
13. A. Vaghani, K. Sood, S. Yu. Security and QoS issues in blockchain enabled next-generation smart logistic networks: A tutorial Blockchain // Research and Applications. 2022. Vol.3. P. 1–14. doi: 10.1016/j.bcra.2022.100082.
14. С. Журавель, О. Шпур, Ю. Пиріг. Досягнення консенсусу в розподілених сервісних системах // Інфокомунікаційні технології та електронна інженерія. 2022. № 2 (4). С. 58–66. doi:10/23939/ictee2022.02.058.
15. В.В. Гнатушенко, Г.К. Витовтов. Аналіз систем масового обслуговування при стрибкоподібній зміні інтенсивностей потоків інформації // Прикладні питання математичного моделювання. 2021. Т. 4, no 2.1. С. 76–83. doi: 10.32782/KNTU2618-0340/2021.4.2.1.7.

Надійшла до редколегії 02.06.2024

Відомості про авторів:

Токар Любов Олександрівна – канд. техн.. наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри інфокомунікаційної інженерії імені В.В. Поповського, Україна; email: liubov.tokar@nure.ua; ORCID: https://orcid.org/0000-0002-7780-1928

Цилюрик Вадим Євгенович - компанія IT-Lance, Україна, email: vadym.tsyliuryk@nure.ua

Солоділов Віктор Васильович – Харківський національний університет радіоелектроніки, магістр кафедри інфокомунікаційної інженерії імені В.В. Поповського, Україна; email: viktor.solodilov@nure.ua