УДК 004.72

А. В. МИРГОРОДСЬКИЙ, О. В. РОМАНЮК, О. Н. РОМАНЮК, Н. В. ТІТОВА

РОЗРОБКА МЕТОДУ ЗАБЕЗПЕЧЕННЯ ВИСОКОЇ ДОСТУПНОСТІ ДЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ УПРАВЛІННЯ КОНФІГУРАЦІЯМИ

Вінницький національний технічний університет, Вінниця, e-mail: psv@vntu.edu.ua
Національний університет Одеська Політехніка», Україна

Анотація. Запропоновано метод забезпечення високої доступності для програмного забезпечення управління конфігураціями. Розглянуто сучасний стан сфери управління електронними ресурсами, наведено причини застосування інструментів автоматизації. Проаналізовано переваги застосування програмного забезпечення для управління конфігураціями, наведено приклади використання підходів Infrastructure as Code та GitOps при автоматизації розгортання та масштабування електронних ресурсів. Проведено аналіз існуючих методів забезпечення високої доступності. Проведено розробку методу забезпечення високої доступності. Результуючий метод забезпечення високої доступності бере за основу алгоритм консенсусу Raft та підхід кластеризації програмної системи і розширює їх додатковими рішеннями. Проведено розробку алгоритму запропонованого методу, детально розглянуто результуючу блок-схему алгоритму та окремі кроки виконання. Проведено оцінку ефективності розробленого методу. Проведено апріорне ранжування ряду факторів, що оцінюють ефективність стратегій та методів автоматичного відновлення. Аналіз результатів показав, що запропонована розробка реалізує в собі найбільш значущі для експертів фактори, а за показниками RTO та RPO запропонований метод може працювати на рівні з існуючими популярними стратегіями аварійного відновлення.

Ключові слова: управління конфігураціями, Infrastructure as Code, метод, розподілені системи, відмовостійкість, висока доступність, аварійне відновлення, стратегії аварійного відновлення.

Abstract. The article proposes its own method of providing high availability for configuration management software. The current state of the electronic resources management sphere was examined, the reasons for the use of automation tools were provided. The advantages of using configuration management software were analyzed, examples of using Infrastructure as Code and GitOps approaches to automate the deployment and scaling of electronic resources were given. The existing methods of ensuring high availability were analyzed. The development of our own method of ensuring high availability was carried out. The resulting method of providing high availability is based on the Raft consensus algorithm and the software system clustering approach and extends them with its own solutions. The algorithm of the proposed method was developed, the resulting flowchart of the algorithm and individual steps of its implementation were described in detail. The efficiency of the developed method was evaluated. An a priori ranking of a number of factors that evaluate the effectiveness of automatic recovery strategies and methods was conducted. The analysis of the results has shown that the proposed method implements the most important factors for experts, and in terms of RTO and RPO, the method can work on a par with existing popular disaster recovery strategies.

Keywords: configuration management, Infrastructure as Code, method, distributed systems, fault tolerance, high availability, disaster recovery, disaster recovery strategies.

DOI: 10.31649/1681-7893-2023-46-2-64-75

ВСТУП

Сучасні цифрові системи, що використовуються кінцевими користувачами у повсякденному житті для виконання роботи, дозвілля та відпочинку, потребують постійної підтримки та контролю. Лічені хвилини простою через аварійну ситуацію з інфраструктурою можуть принести великим IT-компаніям значні фінансові та репутаційні втрати [1].

Тому, надзвичайно важливими є питання підтримки в робочу стані як апаратної, так і програмної складових електронних ресурсів.

Прості скрипти та програми для автоматизації типових завдань добре підходять лише для простих задач, але мають ряд недоліків, таких як складність підтримки та масштабування. Через це виник окремий клас програмного забезпечення, що призначений для управління конфігураціями. Основна мета додатків для управління конфігураціями – це привести надану систему (набір з одного або більше серверів, що можуть виконувати різні ролі та вимагати встановлення і конфігурацію різних компонентів) в бажаний стан [2]. Цей бажаний стан може включати в себе як просту інсталяцію програмних компонент, так і більш складні задачі, таких як динамічна генерація конфігураційних файлів, зміна параметрів ОС або реєстру тощо.

Програмне забезпечення для управління конфігураціями дозволяє швидко перетворити набір нових чистих серверів в повністю функціонуючу систему, що готова до експлуатації, з мінімальними витратами часу та без втручання користувача ПЗ, а стандартизація усіх задач по автоматизації у форматі одного програмного продукту дозволяє зменшити часові та фінансові витрати на підтримку [3].

Проте, сучасні електронні ресурси та обчислювальні системи розвиваються в сторону подальшого зменшення можливого впливу аварійних ситуацій та непрацездатності окремих серверів. Через це поступово з'явилися такі терміни, як висока доступність, горизонтальне масштабування, відмовостійкість тощо. Дедалі більше систем відмовляються від монолітної архітектури та намагаються частково або повністю децентралізувати основні потоки виконання, що дозволяє збільшити продуктивність ПЗ шляхом розподілення задач між декількома окремими машинами, а не за допомогою зміни сервера на більш потужний [4].

Класичне програмне забезпечення для управління конфігураціями погано пристосоване для таких динамічних сценаріїв, а також рідко надає варіанти забезпечення високої доступності – самі сервери, що контролюють процес приведення системи в бажаний стан, залишаються вразливими до будь-яких збоїв та аварійних ситуацій [5]. Саме тому розробка методів і програмних засобів управління конфігураціями для підвищення ефективності процесів розгортання та масштабування електронних ресурсів є досить актуальною задачею.

АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ЗАБЕЗПЕЧЕННЯ ВИСОКОЇ ДОСТУПНОСТІ

Висока доступність – це здатність системи протистояти апаратним і програмним збоям та забезпечувати працездатність і можливість обробки запитів користувачів навіть при відмові частини компонентів. Для забезпечення високої доступності використовують цілий ряд різноманітних підходів:

– кластеризація та надлишковість;
– резервне копіювання та автоматичне відновлення;
– масштабованість;
– балансування навантаження;
– моніторинг та автоматизація типових задач.

Розглянемо більш детально окремі підходи.

Кластеризація (англ. clustering) – це підхід, де використовують декілька фізичних або віртуальних серверів для декомпозиції системи та забезпечення високої доступності системи або окремих її сервісів [6]. Сервери в кластері можуть працювати по-різному в залежності від реалізації режиму високої доступності: це може бути просте дублювання ресурсів або ж повноцінне розподілення навантаження між всіма вузлами кластеру.

При дублюванні зазвичай один сервер виступає в якості головного – він приймає запити і на читання даних, і на їх модифікацію. Решта вузлів або зовсім не беруть участі в обробці запитів, або ж опрацьовують лише операції вичитування даних – таким чином лише головний сервер має можливість змінювати стан системи і містить гарантовано коректну версію всіх даних. Другорядні вузли можуть створюватися шляхом резервного копіювання і відновлення або ж постійною синхронізацією даних між лідером кластера та іншими серверами – такий підхід зазвичай використовується в системах управління базами даних з підтримкою високої доступності. В деяких випадках використовується комбінація обох способів – це залежить лише від обраної стратегії відмовостійкості.

Повноцінне розподілення навантаження передбачає одночасну роботу всіх або частини вузлів кластеру, але це не означає, що відразу декілька машин можуть бути лідерами системи – зазвичай окремі сервери або виконують окремі відведені для них ролі, або ж якась внутрішня чи зовнішня система відповідає за розподіл навантаження [7]. До другого способу також можна відвести безсерверні

обчислення (англ. serverless computing). Через технічні особливості їх механізму роботи неможливо локально зберігати довгострокові дані, тому для цього використовують окремі сервіси для роботи з кешем або ж зовнішні БД. Таким чином, з використанням безсерверних обчислень загальна архітектура системи частково ускладняється, але її масштабування стає набагато простішим.

Крім того, кластеризацію з розподіленням навантаження можна віднести до горизонтального масштабування – підходу до підвищення продуктивності та надійності системи шляхом додавання нових ресурсів. Вертикальне масштабування, з іншого боку, передбачає просте збільшення характеристик серверів, що має фізичні ліміти у вигляді доступних апаратних ресурсів. Горизонтальне масштабування дозволяє значно гнучкіше масштабувати обчислювальну систему, але вимагає відповідну програмну підтримку.

Резервне копіювання та автоматичне відновлення – це важливі процеси для високої доступності, але окремо від інших підходів вони не можуть забезпечити відмовостійкість без повної або часткової недоступності системи. Резервне копіювання будь-яких важливих даних з БД або ж елементів інфраструктури (знімки ОС, файли для IaC та інше) є хорошою практикою, особливо важливою для production-систем і додатків [8]. В комбінації з механізмами автоматичного відновлення можна отримати хороші значення RTO (англ. Recovery Time Objective, цільовий час відновлення – допустимий час простою сервісу в разі збою) та RPO (англ. Recovery Point Objective, цільова точка відновлення – допустимий обсяг втрати даних у разі збою), що дозволять покращити загальну відмовостійкість і високу доступність продукту.

Масштабованість (англ. scalability) – це здатність системи збільшувати власні параметри для обробки зростаючої кількості даних і запитів [9]. Важливою вимогою масштабованості є уникнення будь-яких втрат якості обробки даних – зовнішній користувач при збільшенні навантаження на систему не повинен спостерігати значних змін в її поведінці. Для забезпечення необхідного рівня продуктивності в будь-який момент часу зазвичай використовують горизонтальне або вертикальне масштабування. Більшість хмарних провайдерів мають власні сервіси та інструменти для підтримки горизонтального масштабування, такі як AWS Auto Scaling Groups, Google Cloud Managed Instance Groups, Azure Autoscale та інші.

Балансування навантаження (англ. load balancing) – це ще один підхід, який зазвичай використовується в поєднанні з іншими методиками реалізації високої доступності. Це процес розподілу запитів або іншого типу навантаження між декількома фізичними або віртуальними серверами, точками входу, компонентами системи тощо [10]. Для балансування навантаження може використовуватися як спеціалізоване програмне забезпечення, так і апаратні рішення, які можуть забезпечити ще більшу пропускну здатність системи та працювати на нижчих рівнях мережевої моделі OSI. Типовий приклад використання – це застосувати балансувальник навантаження в якості вхідної точки для кластеру, в якого декілька вузлів одночасно можуть обробляти запити. Сучасні програмні або апаратні рішення вміють не тільки розподіляти запити, а й забезпечувати додаткові перевірки працездатності вузлів (health checks) та підтримку довготривалих сесій (session stickiness, sticky session, session affinity) [11].

Моніторинг та автоматизація типових задач є важливою частиною підтримки в працездатному стані будь-якої системи з високою доступністю. Правильно налаштована система моніторингу дозволить вчасно дізнатися про проблеми з серверами або ж виявити чинники, що впливають на продуктивність їх роботи [12]. Наприклад, збір і аналіз даних про завантаженість вузлів та параметри горизонтального масштабування можуть допомогти оптимізувати програмне забезпечення (знайти більш завантажені компоненти, ліквідувати вузькі місця додатку тощо) та зменшити витрати на сервери в тому випадку, коли попередньо не виявлені проблеми призводять до використання зайвих ресурсів. Інструменти для автоматизації задач можуть керувати додатковими системами сповіщення або ж вирішувати типові задачі на основі зібраних даних від систем моніторингу, звільняючи цим додатковий час для DevOps-інженерів та SRE-спеціалістів.

Розглянуто ряд підходів та методик, що застосовуються для забезпечення високої доступності системи. Реалізація цих методів або їх комбінації та способи застосування можуть сильно відрізнятися в залежності від типу й задач обчислювальної системи або програмного продукту.

Таким чином актуальним є питання розробки нового методу забезпечення високої доступності, який би забезпечував роботу програмного забезпечення для управління конфігураціями в умовах аварійної ситуації або нестабільного мережевого з’єднання. Це дозволило б застосовувати таке ПЗ в більш комплексних сценаріях використання, які повністю або частково не підтримуються класичними рішеннями для управління конфігураціями.

Мета статті – розробка методу забезпечення високої доступності для програмного забезпечення управління конфігураціями.

РОЗРОБКА МЕТОДУ ЗАБЕЗПЕЧЕННЯ ВИСОКОЇ ДОСТУПНОСТІ

Розробка методу включає в себе вирішення ряду задач, а саме забезпечення роботи системи при непрацездатності керуючого сервера, узгодження даних та маршрутизація запитів клієнтів. Перша задача полягає в можливій втраті зв'язку між керуючим та керованими серверами в процесі приведення системи в бажаний стан. Це може бути викликано мережевими проблемами або непрацездатністю апаратного чи програмного забезпечення керуючого сервера. Кількість керованих серверів варіюється від 2-3 до декількох десятків або сотень в залежності від конкретного варіанту використання, тому таку групу серверів можна сприймати в якості кластеру, де кожен вузол виконує свою власну функцію, але відсутні надлишкові вузли й дані для управління кластером (бажаний стан системи) знаходяться на окремій машині – керуючому сервері. Необхідно застосувати децентралізований (або розподілений) режим взаємодії – керуючий сервер більше не застосовується, натомість всі вузли в системі знають одне про одного та можуть в більш вільному режимі обмінюватись повідомленнями в залежності від присвоєної ролі.

В розподіленій програмній системі з'являється нова задача – узгодження даних. Сервери можуть не тільки повністю виходити з ладу через апаратну або програмну помилку, а й просто частково ставати недоступними в мережі з ненадійним з'єднанням. В такому випадку окремий вузол може продовжити ізольовану роботу й накопичити дані про стан системи, що відрізняється від інших вузлів. При відновленні мережевого з'єднання з іншими серверами виникає конфлікт – існує декілька різних варіантів поточного стану кластеру, тому виконання подальших інструкцій може призвести до неочікуваних результатів. Було вирішено застосувати Raft – алгоритм консенсусу, що вперше був опублікований в 2013 році в якості більш простої заміни Paxos. Специфікації Raft описують лише загальний алгоритм та не надають конкретної реалізації, тому його можна взяти за основу для розробки розподілених систем. В основі алгоритму консенсусу лежить скінченний автомат з трьома станами-ролями для вузлів кластера: лідер, послідовник та кандидат [13]. Для визначення актуальності даних (стану системи) застосовується монотонно зростаючий показник – термін або артикул (англ. term). Під час обміну даними між вузлами значення терміну оновлюється з певною періодичністю – якщо в цей час поточний лідер кластеру не може обробити запит через непрацездатність, то починається процес голосування та зміни ролей.

Запропонований метод додатково розширює стандартний алгоритм Raft. По-перше, для збереження консенсусу та коректного оновлення стану системи клієнтам дозволяється працювали лише з поточним лідером кластеру. Це створює додаткове навантаження на один з вузлів, а також залишає відкритим питання розподілення задач між серверами – при управлінні конфігураціями кожен вузол має власні інструкції для приведення системи в бажаний стан. Задачу зв'язку між поточним лідером кластеру та клієнтом неможливо вирішити за допомогою звичайного балансування навантаження, адже балансувальник повинен не просто перенаправити запит до працездатного вузла, але ще й визначити чи цей сервер є поточним лідером. Більш комплексні програмні балансувальники можна налаштувати для такої інтеграції з ПЗ для управління конфігураціями, але це може обмежити деякі варіанти використання через специфіку налаштування та більших вимог до самої інфраструктури.

Інший варіант, який обрано для запропонованого методу забезпечення високої доступності, полягає в застосуванні додаткового механізму обробки запитів. Вузли системи можуть вільно обмінюватися інформацію між собою, а також завжди знають який з серверів є поточним лідером. При отриманні нового запиту вузол-послідовник може перевірити та отримати адресу лідера та виступити в ролі тимчасового проксі-сервера для цього запиту – тобто перенаправити повідомлення до лідера для подальшої обробки. В залежності від застосованого протоколу комунікації механізм перенаправлення може бути реалізований навіть вбудованими засобами (наприклад, API на основі HTTP/REST має такі можливості). В оригінальному Raft основною задачею лідера є не просто обробка вхідних повідомлень, а централізоване оновлення даних на всіх вузлах системи – саме це й дозволяє досягти консенсусу. Базуючись на цьому, програмне забезпечення для управління конфігураціями може мати декілька окремих сервісів для комунікації з користувачем – окремий набір API може бути доступний на всіх вузлах без залежності до їх ролі та виконувати загальні операції для зчитування даних, а інший сервіс буде працювати лише на сервері-лідері та приймати важливі запити по оновленню даних системи. Відповідно, вузли-послідовники будуть перенаправляти до лідера саме ці повідомлення для оновлення.

Для розподілу задач можна застосувати схожий принцип – лідер кластеру реплікує на всі сервери новий стан, але самостійно не розподіляє задачі. Замість цього на кожному вузлі-послідовнику

буде працювати окремий сервіс для аналізу стану при його оновленні. Основна задача такого сервісу – це зчитати нові відомості для поточного сервера з загального масиву даних, порівняти поточний стан вузла з новим та в разі потреби запустити процес приведення системи в бажаний стан на окремо взятому сервері, що й є основною задачею ПЗ для управління конфігураціями.

Таким чином, розроблено вдосконалений метод забезпечення високої доступності ПЗ для управління конфігураціями за допомогою розширення, адаптації та поєднання існуючих підходів. Метод бере за основу алгоритм консенсусу Raft та підхід кластеризації і розширює їх додатковими рішеннями для забезпечення потрібного функціоналу.

РОЗРОБКА АЛГОРИТМУ ТА БЛОК-СХЕМИ МЕТОДУ

На основі запропонованого методу було створено відповідні алгоритм та блок-схему. Алгоритм забезпечення високої доступності описує загальний потік виконання вузла кластера, а саме його запуск та обробку вхідних команд від клієнта з залежністю від поточної ролі сервера.

Блок-схема алгоритму наведена на рисунку 1. Розглянемо більш детально кроки алгоритму.

Крок 1. Початок.

Крок 2. Користувач вводить шлях до файлу конфігурації вузла, в деяких випадках може використовуватися шлях за замовчуванням (наприклад, локальний файл в директорії додатку); значення зберігається в змінній configFilename.

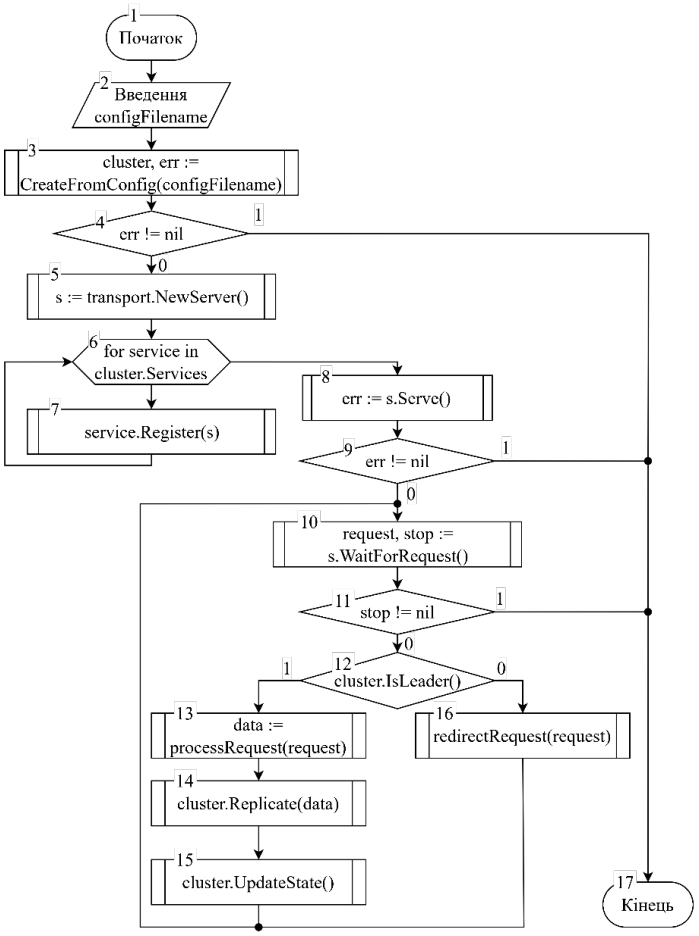

Рисунок 1 – Блок-схема алгоритму забезпечення високої доступності

Крок 3. Відбувається ініціація вузла кластера: функція CreateFromConfig за переданим шляхом зчитує конфігурацію, отримані значення застосовуються для підготовки локального сховища даних та інших процедур.

Крок 4. Якщо на попередньому кроці при ініціації вузла виникла помилка – відбувається перехід до кроку 17. Інакше – перехід до кроку 5.

Крок 5. Відбувається підготовка програмного сервера, який необхідний як для комунікації між вузлами системи, так і для роботи окремих сервісів вузла.

Крок 6. Розпочинається цикл, який перебирає всі сервіси, що необхідні для роботи кластера. Виконання тіла циклу розпочинається на кроці 7, завершення циклу викликає перехід до кроку 8.

Крок 7. Сервіс кластеру реєструється на локальному сервері та ініціює внутрішній стан.

Крок 8. Відбувається запуск сервера та зареєстрованих на ньому сервісів вузла кластера.

Крок 9. Якщо на попередньому кроці при запуску сервера виникла помилка – відбувається перехід до кроку 17. Інакше – перехід до кроку 10.

Крок 10. Сервер очікує нових вхідних повідомлень від клієнта або іншого вузла. Виконання продовжується, коли отримано новий запит request або ж керуючий сигнал для зупинки stop.

Крок 11. Якщо отримано керуючий сигнал stop, то відбувається перехід до кроку 17. Інакше – перехід до кроку 12.

Крок 12. Якщо поточний вузол являється лідером кластера, то відбувається перехід до кроку 13. Інакше – перехід до кроку 16.

Крок 13. Відбувається обробка запиту та отримання даних для реплікації.

Крок 14. Поточний лідер кластера реплікує нові дані стану на інші сервери.

Крок 15. Відбувається процедура порівняння попереднього стану та оновленого; за потребою виконуються додаткові дії для досягнення бажаного стану системи, тобто запускається основний функціонал ПЗ для управління конфігураціями.

Крок 16. Вхідний запит перенаправляється з вузла-послідовника до вузла-лідера.

Крок 17. Кінець виконання.

Таким чином, розроблений алгоритм реалізує в собі усе необхідні частини запропонованого методу забезпечення високої доступності для програмного забезпечення управління конфігураціями.

ОЦІНКА ЕФЕКТИВНОСТІ РОЗРОБЛЕНОГО МЕТОДУ

Одна з властивостей високої доступності – це забезпечення безвідмовної роботи в аварійних ситуаціях, тому варто порівняти розроблений метод з іншими стратегіями аварійного відновлення.

Як вже розглядалося раніше, RTO визначає час на відновлення працездатності системи, а RPO – об’єм даних, який неможливо відновити через особливості тієї чи іншої стратегії відновлення. Іншими словами, RTO та RPO визначаються як час простою та втрати даних відповідно.

Конкретні значення цих показників сильно залежать від цифрової інфраструктури продукту, а також від масштабів розгорнутої системи. Крім того, кількість ресурсів та часу, що необхідні для реалізації тієї чи іншої стратегії відновлення, також відрізняються. Тому, у відкритому доступі складно знайти усереднені статистичні показники RTO/RPO – вони індивідуальні й залежать від конкретного продукту та готовності бізнесу інвестувати в аварійне відновлення.

Проте, є чіткі співвідношення між стратегіями відновлення, які дозволяють оцінити їх показники відносно одне одного. Найбільш популярними стратегіями аварійного відновлення є: «Backup and restore», «Pilot light», «Warm standby», «Multi-site active/active».

Запропонований удосконалений метод забезпечення високої доступності за своїми характеристиками подібний до стратегій «Pilot light» та «Warm standby». Це дві схожі стратегії, що відрізняються кількістю попередньо розгорнутих ресурсів. «Pilot light» передбачає дублювання лише найважливішої інфраструктури, для якої регулярно проводиться реплікація даних, а решта вузлів або сервісів розгортаються лише у випадку аварійної ситуації. «Warm standby» передбачає повне дублювання системи та необхідної інфраструктури, але в зменшених масштабах – ця стратегія добре підходить для систем, що підтримують автоматичне масштабування. У випадку аварійної ситуації додаткові ресурси можуть бути створені відповідно до нового клієнтського навантаження. Ці дві стратегії мають схожі показники та зазвичай обходяться дорожче за «Backup and restore», але й показують кращі показники RTO та RPO – в середньому це десятки хвилин для «Pilot light» та лічені хвилини для «Warm standby».

Для використання запропонованого методу також необхідні додаткові сервери-вузли, які вже в залежності від призначення системи можуть або виконувати певну корисну роботу, або ж просто реплікувати критично важливі дані та очікувати. Тому, вартість застосування запропонованого методу в якості стратегії аварійного відновлення може бути близькою до застосування «Pilot light» або «Warm standby», але показники RTO та RPO повинні бути кращими.

Реплікація даних відбувається в момент обробки нового запиту на оновлення стану кластеру – лідер виконує зміни в своїй власній копії даних, а потім розсилає ці оновлення на інші вузли. Тільки після підтвердження про успішну реплікацію від більшості вузлів лідер зберігає всі ці зміни в якості постійних. Втрата даних може виникнути лише під час обробки нового запиту, коли лідер почав працювати з новими даними, але потрапив в аварійну ситуацію до закінчення реплікації даних. Відповідно, показник RPO в такому сценарії повинен становити не більше декількох секунд для малих об'ємів даних (наприклад, коли в загальному стані є лише базова інформація для підтримки системи в робочому стані) та не більше декількох хвилин для великих об'ємів (наприклад, коли реплікований стан використовують в якості бази даних).

Показник RTO не повинен перевищувати декількох десятків секунд – вузли регулярно комунікують з лідером для перевірки наявності нової інформації, тому виявлення непрацюючого лідера та його заміна на інший вузол відбувається швидко. Так як останні дані про стан системи регулярно реплікуються, то більшість вузлів можуть стати потенційними кандидатами для нового лідера – процес переключення включає в себе лише голосування за конкретний вузол та розсилання оновлення з даними нового лідера.

Для більш об'єктивної оцінки розробленого методу було застосовано апріорне ранжування. Було сформовано ряд факторів, що присутні в розглянутих стратегіях аварійного відновлення. Група з 5-ти експертів, що мають досвід в сфері DevOps, провели ранжування цих факторів. За отриманими даними можна визначити, які показники є найбільш значущими та перевірити наскільки вони реалізовані в запропонованій розробці, щоб зрозуміти її актуальність.

Було обрано наступні фактори для ранжування:

– складність реалізації та підтримки;

– час простою в аварійній ситуації;

– частота реплікації даних або створення резервних копій;

– можливість додаткового використання резервних вузлів;

– автоматизація процесу відновлення.

Оцінки експертів та результати розрахунків наведено в таблиці 1. Після виставлення оцінок було обчислено суму рангів для кожного фактору за формулою:

$$\Delta_k = \sum_{m=1}^{m} a_{km} ,$$

де m – кількість експертів (у випадку даного дослідження – це 5 осіб), а k – кількість факторів, що були використані при ранжуванні (в даному випадку – 5).

Приклад розрахунку для першого фактору:

$$\Delta_1 = 2+5+5+4+4 = 20 .$$

Після цього було розраховано загальну суму рангів та середнє значення суми:

$$\overline{\Delta} = \frac{\sum_{k=1}^{k} \Delta_k}{k} = 20+9+12+22+12 = \frac{75}{5} = 15 .$$

Далі розраховано відхилення суми рангів від середньої суми рангів для кожного фактору:

$$\Delta_k^{\sigma} = \Delta_k - \overline{\Delta} .$$

Приклад розрахунку для першого фактору:

$$\Delta_1^{\sigma} = 20-15 = 5 .$$

Ці дані дозволяють розрахувати коефіцієнт згоди між експертами – занадто мале значення означає, що обрана група має занадто різнобічні погляди й не підходить для ранжування факторів. Спочатку було знайдено суму квадратів відхилення:

$$\mathrm{S} = \sum_{k=1}^{k} \left(\Delta_k^{\sigma}\right)^2 = 25+36+9+49+9 = 128 .$$

І з використанням отриманої суми розраховано коефіцієнт узгодженості:

$$\mathrm{W} = \frac{12\mathrm{S}}{m^2\left(k^3-k\right)} = \frac{12\cdot 128}{5^2\left(5^3-5\right)} = 0{,}512 .$$

Якщо значення коефіцієнту складає більше, ніж 0,5, як вийшло при розрахунках з даною командою та набором факторів, то можна вважати, що в оцінках експертів є певний рівень згоди.

Крім того, отримані дані можна використати для розрахунку критерія узгодженості Пірсона. Він застосовується для перевірки, чи отримана гіпотеза про ранжування факторів є випадковим співпадінням думок експертів, чи вона має певне підґрунтя. Може виникнути ситуація, коли оцінки експертів підпадають під закон розподілу ймовірностей і в такому випадку неможна довіряти отриманому значенню коефіцієнта узгодженості.

Для обчислення використано таку формулу:

$$\chi^2 = \mathrm{W} \cdot m(k-1) = 0,512 \cdot 5(5-1) = 10,24,$$

де $(k-1)$ визначає число ступенів свободи. Табличне значення критерію Пірсона за умови наявності чотирьох ступенів свободи та рівня значимості 0,05 становить 9,5 [14], що менше отриманого значення. Це дозволяє прийняти гіпотезу про те, що співпадіння думок експертів не є випадковим на заданому рівні значимості.

Отримані значення Δ_k дозволяють провести ранжування факторів. Місця розподіляються по мірі зростання суми рангів, тобто перше місце відповідає найменшій сумі.

На основі місць можна розрахувати питому вагу кожного фактору – це дозволяє порівняти фактори відносно одне одного й визначити їх значущість на думку експертів. Розрахунок питомої ваги виконано за допомогою наступної формули:

$$q_k = \frac{2(k - \mathrm{M} + 1)}{k(k+1)},$$

де M – це місце, що отримав фактор на основі ранжування. Всі результати розрахунків наведено в таблиці 1.

Таблиця 1

Вхідні дані та результати апріорного ранжування

Номер	Фактор	Експерти					Δ_k	Δ_k^σ	$\left(\Delta_k^\sigma\right)^2$	M	q_k
		1	2	3	4	5					
1	Складність реалізації та підтримки	2	5	5	4	4	20	5	25	4	0,13
2	Час простою в аварійній ситуації	3	2	1	1	2	9	-6	36	1	0,33
3	Частота реплікації даних	1	3	2	3	3	12	-3	9	2	0,27
4	Додаткове використання вузлів	5	4	3	5	5	22	7	49	5	0,07
5	Автоматизація процесу відновлення	4	1	4	2	1	12	-3	9	3	0,2
Сума		15	15	15	15	15	75	-	128	-	1

Отримані дані дозволяють побудувати апріорну діаграму рангів, яка візуально показує найбільш важливі фактори з точки зору експертів при виборі методу або стратегії для забезпечення високої доступності. Фактори, чия сума рангів менша за середню, можна вважати більш важливими за решту. Отримана діаграма наведена на рисунку 2.

Згідно діаграми, найбільш важливими факторами при виборі методу або стратегії аварійного відновлення є час простою в аварійній ситуації, частота реплікації даних та автоматизація процесу відновлення. Їх сумарна питома вага становить 0,8.

Отримане ранжування дозволяє оцінити наскільки актуальним є запропонований метод забезпечення високої доступності відповідно до вимог і оцінок експертів. Розроблений метод має хороші показники часу простою та частоти реплікації даних – в залежності від ситуації ці параметри можуть збігатися або бути кращими за використання стратегій «Pilot light» та «Warm standby».

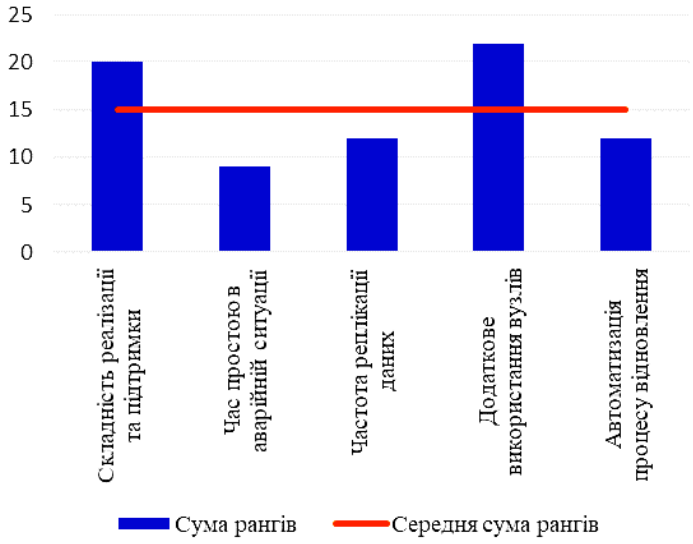

Рисунок 2 – Апріорна діаграма рангів

Процес відновлення працездатності системи у розробленому методі повністю автономний – вузли самостійно перевіряють можливість з'єднання з поточним лідером та ініціюють його зміну в випадку виникнення проблем. Крім того, метод надає можливість використовувати вузли-послідовники для самого процесу управління конфігураціями, а не просто використовує їх в якості пасивних реплік, тому сервери мають можливість виконувати додаткове корисне навантаження. Проте, цей фактор експерти оцінили в якості найменш важливого. Складність реалізації та підтримки не розглядається з аналогічної причини – цей фактор посідає четверте місце в ранжуванні і його оцінка може сильно варіюватися в залежності від варіантів застосування того чи іншого методу або стратегії для реальної системи.

ВИСНОВКИ

У статті розглянуто розробку запропонованого методу забезпечення високої доступності для програмного забезпечення управління конфігураціями. Апріорне ранжування показало, що метод реалізує найбільш важливі за оцінками експертів фактори, а за показниками RTO та RPO він показує себе краще за деякі інші популярні стратегії аварійного відновлення. Розроблений метод забезпечення високої доступності для програмного забезпечення управління конфігураціями є актуальним та доцільним для використання.

СПИСОК ЛІТЕРАТУРИ

1. Franke, U., (2020). IT service outage cost: case study and implications for cyber insurance. The Geneva Papers on Risk and Insurance - Issues and Practice [online]. 45(4), 760–784. [Viewed 11 November 2023]. Available from: doi: 10.1057/s41288-020-00177-4
2. Olawuyi, J., Benson-Emenike, M. and Onuoha, O., (2023). Configuration management. West Africa Journal of Science, Technology and Social. 97–105.
3. Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J. and Kudlacek, M., (2018). Unleashing full potential of ansible framework: university labs administration. In: 2018 22nd conference of open innovations association (FRUCT), 15–18 May 2018, Jyvaskyla [online]. IEEE. [Viewed 11 November 2023]. Available from: doi: 10.23919/fruct.2018.8468270
4. Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E. and Toulkeridis, T., (2020). From monolithic systems to microservices: a comparative study of performance. Applied Sciences [online]. 10(17), 5797. [Viewed 11 November 2023]. Available from: doi: 10.3390/app10175797
5. Миргородський, А. В. та Романюк, О. В., (2022). Аналіз методів для управління конфігураціями при розгортанні електронних ресурсів. У: Електронні інформаційні ресурси: створення, використання, доступ, 28–29 листопада 2022, Суми/Вінниця, Україна. Суми/Вінниця: НІКО/ВНТУ. с. 152–156.
6. Barman, S., Gope, H. L., Manjurul Islam, M. M., Hasan, M. M. and Salma, U., (2016). Clustering techniques for software engineering. Indonesian Journal of Electrical Engineering and Computer

Science [online]. 4(2), 465. [Viewed 11 November 2023]. Available from: doi: 10.11591/ijeecs.v4.i2.pp465-472
7. Nwobodo, I., (2015). Cloud computing: a detailed relationship to grid and cluster computing. International Journal of Future Computer and Communication [online]. 4(2), 82–87. [Viewed 11 November 2023]. Available from: doi: 10.7763/ijfcc.2015.v4.361
8. Andry, J. F. and Po, H., (2017). Using backup and restore automation from disaster in university information systems. In: 2nd international conference on innovative research across disciplines (ICIRAD 2017), 26 August 2017, Denpasar, Bali-Indonesia [online]. Paris, France: Atlantis Press. [Viewed 11 November 2023]. Available from: doi: 10.2991/icirad-17.2017.1
9. Bondi, A. B., (2000). Characteristics of scalability and their impact on performance. In: The second international workshop, Ottawa, Ontario, Canada [online]. New York, New York, USA: ACM Press. [Viewed 11 November 2023]. Available from: doi: 10.1145/350391.350432
10. Rajak, R., Choudhary, A. and Sajid, M., (2023). Load balancing techniques in cloud platform: a systematic study. International Journal of Experimental Research and Review [online]. 30, 15–24. [Viewed 11 November 2023]. Available from: doi: 10.52756/ijerr.2023.v30.002
11. Stecca, M., Bazzucco, L. and Maresca, M., (2011). Sticky session support in auto scaling iaas systems. In: 2011 IEEE world congress on services (SERVICES), 4–9 July 2011, Washington, DC, USA [online]. IEEE. [Viewed 11 November 2023]. Available from: doi: 10.1109/services.2011.27
12. Gaol, F. L., Santoso, S. and Matsuo, T., (2022). Design and development of the application monitoring the use of server resources for server maintenance. Open Engineering [online]. 12(1), 524–538. [Viewed 11 November 2023]. Available from: doi: 10.1515/eng-2022-0055
13. Миргородський, А. В. та Романюк, О. В., (2023). Розробка розподілених систем з використанням алгоритму консенсусу raft. У: LII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, 21–23 червня 2023, Вінниця, Україна [онлайн]. Вінниця: ВНТУ. [Дата звернення 10 листопада 2023]. Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023/paper/view/17164
14. Singhal, R. and Rana, R., (2015). Chi-square test and its application in hypothesis testing. Journal of the Practice of Cardiovascular Sciences [online]. 1(1), 69. [Viewed 9 November 2023]. Available from: doi: 10.4103/2395-5414.157577.
15. Olexander N. Romanyuk, Sergii V. Pavlov, and etc. "A function-based approach to real-time visualization using graphics processing units", Proc. SPIE 11581, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2020, 115810E (14 October 2020); https://doi.org/10.1117/12.2580212.
16. Leonid I. Timchenko, Natalia I. Kokriatskaia, Sergii V. Pavlov, and etc. "Q-processors for real-time image processing", Proc. SPIE 11581, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2020, 115810F (14 October 2020); https://doi.org/10.1117/12.2580230
17. Olexander N. Romanyuk, Sergii V. Pavlov, and etc. "Transformation of polygonal description of objects into functional specification based on three-dimensional patches of free forms", Proc. SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, 1117622 (6 November 2019); https://doi.org/10.1117/12.2537043.
18. Pavlov S.V. Selective irradiation of superficial tumours depending on the photosensitiser fluorescence in the tissue/ O.M. Chepurna, V.V. Kholin, S.V. Pavlov, and etc. // Information Technology in Medical Diagnostics II. CRC Press, Balkema book, 2019 Taylor & Francis Group, London, UK, PP. 53-58; https://www.taylorfrancis.com/books/e/9780429615498/ chapters/10.1201/9780429057618-8.

REFERENCES

1. Franke, U., (2020). IT service outage cost: case study and implications for cyber insurance. The Geneva Papers on Risk and Insurance - Issues and Practice [online]. 45(4), 760–784. [Viewed 11 November 2023]. Available from: doi: 10.1057/s41288-020-00177-4
2. Olawuyi, J., Benson-Emenike, M. and Onuoha, O., (2023). Configuration management. West Africa Journal of Science, Technology and Social. 97–105.
3. Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J. and Kudlacek, M., (2018). Unleashing full potential of ansible framework: university labs administration. In: 2018 22nd conference of open innovations association (FRUCT), 15–18 May 2018, Jyvaskyla [online]. IEEE. [Viewed 11 November 2023]. Available from: doi: 10.23919/fruct.2018.8468270
4. Tapia, F., Mora, M. Á., Fuertes, W., Aules, H., Flores, E. and Toulkeridis, T., (2020). From monolithic systems to microservices: a comparative study of performance. Applied Sciences [online].

10(17), 5797. [Viewed 11 November 2023]. Available from: doi: 10.3390/app10175797
5. Myrhorodskyi, A.V. and Romanyuk, O.V., (2022). Analysis of methods for configuration management when deploying electronic resources. In: Electronic Information Resources: Creation, Use, Access, November 28-29, 2022, Sumy/Vinnytsia, Ukraine. Sumy/Vinnytsia: NIKO/VNTU. with. 152–156.
6. Barman, S., Gope, H. L., Manjurul Islam, M. M., Hasan, M. M. and Salma, U., (2016). Clustering techniques for software engineering. Indonesian Journal of Electrical Engineering and Computer Science [online]. 4(2), 465. [Viewed 11 November 2023]. Available from: doi: 10.11591/ijeecs.v4.i2.pp465-472
7. Nwobodo, I., (2015). Cloud computing: a detailed relationship to grid and cluster computing. International Journal of Future Computer and Communication [online]. 4(2), 82–87. [Viewed 11 November 2023]. Available from: doi: 10.7763/ijfcc.2015.v4.361
8. Andry, J. F. and Po, H., (2017). Using backup and restore automation from disaster in university information systems. In: 2nd international conference on innovative research across disciplines (ICIRAD 2017), 26 August 2017, Denpasar, Bali-Indonesia [online]. Paris, France: Atlantis Press. [Viewed 11 November 2023]. Available from: doi: 10.2991/icirad-17.2017.1
9. Bondi, A. B., (2000). Characteristics of scalability and their impact on performance. In: The second international workshop, Ottawa, Ontario, Canada [online]. New York, New York, USA: ACM Press. [Viewed 11 November 2023]. Available from: doi: 10.1145/350391.350432
10. Rajak, R., Choudhary, A. and Sajid, M., (2023). Load balancing techniques in cloud platform: a systematic study. International Journal of Experimental Research and Review [online]. 30, 15–24. [Viewed 11 November 2023]. Available from: doi: 10.52756/ijerr.2023.v30.002
11. Stecca, M., Bazzucco, L. and Maresca, M., (2011). Sticky session support in auto scaling iaas systems. In: 2011 IEEE world congress on services (SERVICES), 4–9 July 2011, Washington, DC, USA [online]. IEEE. [Viewed 11 November 2023]. Available from: doi: 10.1109/services.2011.27
12. Gaol, F. L., Santoso, S. and Matsuo, T., (2022). Design and development of the application monitoring the use of server resources for server maintenance. Open Engineering [online]. 12(1), 524–538. [Viewed 11 November 2023]. Available from: doi: 10.1515/eng-2022-0055
13. Myrhorodskyi, A.V. and Romanyuk, O.V., (2023). Development of distributed systems using the raft consensus algorithm. In: LII Scientific and Technical Conference of the Faculty of Information Technologies and Computer Engineering, June 21–23, 2023, Vinnytsia, Ukraine [online]. Vinnytsia: VNTU. [Date of application November 10, 2023]. Access mode: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023/paper/view/17164
14. Singhal, R. and Rana, R., (2015). Chi-square test and its application in hypothesis testing. Journal of the Practice of Cardiovascular Sciences [online]. 1(1), 69. [Viewed 9 November 2023]. Available from: doi: 10.4103/2395-5414.157577.
15. Olexander N. Romanyuk, Sergii V. Pavlov, and etc. "A function-based approach to real-time visualization using graphics processing units", Proc. SPIE 11581, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2020, 115810E (14 October 2020); https://doi.org/10.1117/12.2580212.
16. Leonid I. Timchenko, Natalia I. Kokriatskaia, Sergii V. Pavlov, and etc. "Q-processors for real-time image processing", Proc. SPIE 11581, Photonics Applications in Astronomy, Communications, Industry, and High Energy Physics Experiments 2020, 115810F (14 October 2020); https://doi.org/10.1117/12.2580230
17. Olexander N. Romanyuk, Sergii V. Pavlov, and etc. "Transformation of polygonal description of objects into functional specification based on three-dimensional patches of free forms", Proc. SPIE 11176, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019, 1117622 (6 November 2019); https://doi.org/10.1117/12.2537043.
18. Pavlov S.V. Selective irradiation of superficial tumours depending on the photosensitiser fluorescence in the tissue/ O.M. Chepurna, V.V. Kholin, S.V. Pavlov, and etc. // Information Technology in Medical Diagnostics II. CRC Press, Balkema book, 2019 Taylor & Francis Group, London, UK, PP. 53-58; https://www.taylorfrancis.com/books/e/9780429615498/ chapters/10.1201/9780429057618-8.

Надійшла до редакції 8.10.2023 р.

МИРГОРОДСЬКИЙ АНДРІЙ ВІКТОРОВИЧ – студент, Вінницький національний технічний університет, Вінниця, Україна, ***e-mail: mirgorodskijav@gmail.com***

РОМАНЮК ОКСАНА ВОЛОДИМИРІВНА – к.т.н., доцент кафедри програмного забезпечення, Вінницький національний технічний університет, Вінниця, Україна,
e-mail: romaniukoksanav@gmail.com

РОМАНЮК ОЛЕКСАНДР НИКИФОРОВИЧ – д.т.н., професор кафедри програмного забезпечення, Вінницький національний технічний університет, Вінниця, Україна, ***e-mail: rom8591@gmail.com***

ТІТОВА НАТАЛІЯ ВОЛОДИМИРІВНА – д.т.н., професор, завідувачка кафедри біомедичної інженерії, *Національний університет «Одеська політехніка»,* ***e-mail: tnv.titova@gmail.com***

A. V. MYRHODSKYY, O. V. ROMANYUK, O. N. ROMANYUK, N. V. TITOVA

DEVELOPMENT OF A HIGH AVAILABILITY METHOD FOR CONFIGURATION MANAGEMENT SOFTWARE

Vinnytsia National Technical University
National University “Odessa Polytechnika”