DOI 10.31891/2307-5732-2021-297-3-54-58
УДК 004.052.3

Д.Ю. АНТІЧ, Г.І. РАДЕЛЬЧУК
Хмельницький національний університет

УДОСКОНАЛЕННЯ АЛГОРИТМІВ ЗАБЕЗПЕЧЕННЯ ВІДМОВОСТІЙКОСТІ ПРОГРАМНИХ СИСТЕМ

У роботі представлено концепції відмовостійкості програмних систем та методи реагування системи на відмову. У дослідженні вдосконалено метод реагування системи на відмову шляхом проєктування комплексного рішення, що включає в себе доопрацювання та розширення класичних патернів відмовостійкості. Обґрунтовано доцільність та актуальність проектування нового методу відмовостійкості. Удосконалено алгоритми автоматичного реагування та попередження відмов, що дозволяє зменшити кількість помилок, які виникають у системі, у порівнянні з існуючими рішеннями. Результатом дослідження є покращений метод забезпечення відмовостійкості програмних систем.

Ключові слова: програмна система, алгоритм, відмовостійкість, патерн відмовостійкості

DMYTRO YURIIOVYCH ANTICH, GALINA IVANIVNA RADELCHUK
Khmelnytsky National University

IMPROVEMENT OF SOFTWARE SYSTEMS FAULT TOLERANCE ENSURING ALGORITHMS

The study investigates the concepts of fault tolerance and methods of system responses to failures. The study is based on the research of modern resiliency patterns and common approaches of reaction to failures. During the research, the common unresolved issues with modern resiliency and fault tolerance approaches were defined. The study improved the method of the system response to failures by designing a comprehensive solution that includes refinement and expansion of classical patterns of fault tolerance as a proposal to resolve common problems. The new solution of fault tolerance is based on the combination of basic monitoring approaches, load balancing approaches, circuit breaker pattern, and re-designing of the sharding pattern to be applicable not only for databases but also for modern applications. The new solution is based on an automatic decision-making expert system, which based on anonymous data saved by the monitoring layer decides the root cause of the issue and validates which scenario is applicable for the current situation. Based on the decision system can either enable a user and load balancing approaches by isolating harmful users using improved sharding and load-balancing solutions or enable a circuit breaker to temporarily disable the faulty features. The new method of resiliency is supposed to prevent and reduce more errors compared to the existing solutions in the domain of fault tolerance and resiliency, thus the efficiency of the new approach is higher. The expediency and urgency of designing a new method of fault tolerance are substantiated by expressing the importance of resolving existing problems. Improved methods of automatic response and failure prevention, which allowed to reduce the number of errors that occur in the system compared to existing solutions in resiliency and fault tolerance.

Keywords: software system, algorithm, fault tolerance, fault tolerance pattern

Вступ. Постановка проблеми

У сучасному світі мережа Інтернет займає значну частину життя людини. Кожен з нас щодня перевіряє інформацію про погоду, купує товари, спілкується з друзями чи знайомими, читає новини, книги, переглядає відео онлайн тощо. За час пандемії більшість звичних нам речей змушена була адаптуватись до нової реальності, зокрема, значна кількість видів діяльності почала здійснюватись онлайн. Наприклад, багато компаній змушені були перевести співробітників на віддалену роботу, університети та школи перейшли у режим дистанційного навчання тощо.

Таким чином, у зв'язку зі стрімкою діджиталізацією виникає потреба забезпечити надійність та відмовостійкість програмних систем (ПС). А це, у свою чергу, породжує потребу в алгоритмах відмовостійкості, які нададуть можливість підвищити працездатність ПС та забезпечити їх швидке відновлення після настання відмов.

Аналіз останніх досліджень та публікацій

У результаті досліджень університету Карнегі-Меллона (США) визначено поняття відмовостійкості та доведено, що жодна програмна система не є на 100% відмовостійкою або не є стійкою до відмов взагалі [1]. Група дослідників [2] визначила основні фактори, що шкодять відмовостійкості, та який вплив ці фактори мають на ПС. Такими факторами є наступні: розмір та складність програмної системи; взаємозалежність та взаємозв'язок; мережева орієнтованість; стрімка глобалізація компанії; програмне забезпечення з відкритим вихідним кодом; гібридизація; швидкі зміни; повторне використання у різних контекстах.

Комплексний підхід до відмовостійкості програмних систем включає в себе чотири базові методи реалізації відмовостійкості, а саме:

- реакція на помилки (як тільки виникає помилка, програмна система автоматично або розробники програми у ручному режимі повинні відреагувати на помилки);
- автоматичне логування (програмна система повинна автоматично логувати всі проблеми та помилки для можливості автоматичного реагування на проблеми);
- метрики MTBF, MTTF, MTTR (ці метрики визначають середній час між відмовами, середній час до відмови, середній час на усунення відмови);
- автоматизація плану відновлення (необхідно автоматизувати кроки, які система має здійснити для відновлення нормального функціонування).

Виділення невирішених частин загальної проблеми

Проаналізувавши описані у роботі [3] стратегії управління та запобігання потенційним помилкам, можна виділити наступний алгоритм реакції системи на відмову:

- автоматичне визначення помилки – система повинна самостійно виявити помилку, зберегти додаткову інформацію (ланцюжок виклику, подію, що спричинила помилку, тощо) та, за можливості, вивести користувачу наперед визначене повідомлення про помилку;
- аналіз та пояснення помилки (на цьому етапі необхідно висунути гіпотезу про причину помилки та знайти спосіб її вирішення);
- обробка помилок програмною системою за допомогою використання патернів відмовостійкості, що підходять для конкретної проблеми.

Найчастіше використовуються наступні методи реалізації відмовостійкості ПС:

- шаблон вимикача;
- шаблон повторень;
- шаблон компенсації транзакцій;
- шардінг.

Шаблон вимикача обробляє несправності, для вирішення яких може знадобитися різний час при підключенні до віддаленого сервісу чи ресурсу. Вимикач діє як проксі-сервер для операцій, які можуть вийти з ладу. Проксі відстежує кількість останніх помилок, які сталися, і використовує цю інформацію для прийняття рішення – дозволити продовження операції чи перервати її та повернути помилку [4].

Шаблон повторень реалізує програму для обробки збоїв, коли система намагається здійснити запит до сервісу чи ресурсу, шляхом прозорого повторення невдалої операції. Якщо програма виявляє помилку під час спроби надіслати запит віддаленому сервісу, вона може впоратися з помилкою, використовуючи такі стратегії:

- скасувати запит (якщо помилка вказує на те, що несправність не є тимчасовою або запит навряд чи буде успішним при його повторенні, то програма повинна скасувати операцію та повідомити про помилку);
- повторити спробу (якщо конкретне повідомлення про помилку є незвичним чи рідкісним, то це може бути спричинено незвичними обставинами, такими, наприклад, як пошкодження мережевого пакета під час його передачі; у цьому випадку програма може негайно повторити невдалий запит, оскільки та сама помилка навряд чи буде повторена, і запит, ймовірно, буде успішним);
- повторити спробу після затримки (якщо несправність спричинена проблемами з функціонуванням мережі або надмірним навантаженням на додаток, то може знадобитися деякий період часу, поки проблеми з підключенням будуть виправлені або всі поточні запити оброблені); програма повинна почекати деякий час перед повторною спробою запиту [5].

Шаблон компенсації транзакцій скасовує роботу, виконану за допомогою декількох кроків, які в сукупності визначають узгоджену в кінцевому рахунку операцію, якщо один або кілька кроків завершилися з помилкою. Таким чином, цей шаблон дає змогу повернутись до попереднього стабільного стану програмної системи та уникнути розбіжностей в даних. Цей патерн є особливо актуальним у випадках, коли деяка дія потребує запису в декілька таблиць бази даних чи виклику декількох сервісів [6].

Шардінг реалізує розбиття таблиць у базі даних на менші «шматки» (шарди). Розбиття може бути як горизонтальним, так і вертикальним. При вертикальному розбитті стовпці таблиці зберігаються в окремій базі даних, а при горизонтальному – рядки однієї таблиці зберігаються в декількох вузлах бази даних [7].

Таким чином, стратегія відмовостійкості не передбачає проактивну реакцію на потенційні відмови ПС, а працює з уже існуючими. Тому вказана предметна область має наступні невирішені проблеми:

- відсутність алгоритму для передбачення відмови, що призводить до відсутності автоматичного блокування першопричини потенційної відмови;
- відсутність реакцій на часткові відмови та підходів для автоматичного вирішення часткової непрацездатності системи.

Загалом, методи та алгоритми відмовостійкості на сьогоднішній день фокусуються на тому, щоб зменшити негативний вплив відмов на ПС після того, як система почала працювати некоректно. Однак, методи та підходи, що використовуються у сучасних програмних продуктах, не реагують на часткові відмови, не здатні автоматично визначати причину часткової відмови та здійснювати автоматичні спроби уснути таку відмову або основну причину відмови.

Формулювання цілей

Для проведення дослідження сформовано наступні цілі: провести теоретичний аналіз способів та методів забезпечення відмовостійкості програмних систем; охарактеризувати основні поняття та проблеми відмовостійкості; описати існуючі методи відмовостійкості; виділити наявні проблеми у галузі відмовостійкості ПС та описати шляхи їх вирішення; удосконалити існуючі алгоритми забезпечення відмовостійкості ПС.

Виклад основного матеріалу

На основі аналізу невирішених проблем відмовостійкості програмних систем було запропоновано розробити власний алгоритм шляхом комбінації та модифікації існуючих методів, що дозволить оптимізувати наявні вузькі місця в існуючих підходах. Визначено вимоги до удосконаленого алгоритму

відмовостійкості ПС:

- алгоритм повинен реагувати на часткові відмови, а саме: автоматично аналізувати та сканувати усі помилки, що з'являються у програмній системі;
- алгоритм повинен мати можливість автоматично висувати гіпотези про причину помилок та намагатись автоматично усувати помилку.

Для початку було вирішено використати підходи до моніторингу та збереження даних задля того, щоб алгоритм мав змогу аналізувати вхідні дані користувачів та робити припущення про можливі потенційні відмови системи.

Для вирішення поставленої задачі необхідно зберігати дані про користувачів та їх запити, тому удосконалений метод відмовостійкості збиратиме та аналізуватиме дані, які представлені у таблиці 1.

Таблиця 1

Вхідні дані алгоритму

Вхідний параметр	**Приклад даних**
IP-адреса користувача	188.123.19.208
Операційна система користувача	macOS 10.15.7
Браузер користувача	Chrome 91.0.4472.106
Запит, що здійснюється	somesite.com/login

Вхідні параметри запиту	{
	"username": "dmytro",
	"password": "sometext"
	}

Відповідь системи на запит користувача	{
	"code": 300,
	"message": "Login failed"
	"meta info": "Uncaught PHP Exception Ramsey\Uuid\Exception\InvalidUuidStringException: "Invalid UUID string: " at /var/www/project/vendor/ramsey/uuid/src/Codec/StringCodec.php line 146"
	}

На основі даних користувача формується унікальний анонімний хеш-ключ, який слугує ідентифікатором користувача. Аналізуючи дані, алгоритм може автоматично припустити, де існує проблема, та спробувати її автоматично вирішити.

Після завершення роботи моніторингу запускається логіка припущень. Для цього реалізовано алгоритм припущень, який на основі даних, збережених моніторингом, чи використовуючи балансувач та шардінг, ізолює потенційно небезпечних користувачів або виконує запуск патерну вимикача.

У разі, якщо система буде надсилати помилки всім користувачам, які виконують певний запит, то тоді алгоритм здійснить припущення, що проблема є локальною та стосується лише одного сервісу (чи однієї кінцевої точки). В результаті алгоритм автоматично вимкне даний сервіс за допомогою патерна вимикача. Наприклад, якщо сторінка авторизації не працює, користувачі здійснюють запити на авторизацію, проте отримують помилки, тоді алгоритм аналізує помилки і дані користувачів та робить припущення, що проблема стосується конкретного модуля системи і блокує всі подальші запити до вирішення проблеми.

Схематичне відображення роботи алгоритму у такому випадку представлено на рисунку 1.

На рисунку 2 представлена робота алгоритму у випадку повної працездатності ПС.

Описаний підхід дасть можливість ізолювати проблемні частини програмного продукту, а це, у свою чергу, дасть можливість користувачам продовжувати користуватись частиною програми, яка для них залишається працездатною.

Рис. 1. Робота алгоритму вимикача – відкритий стан

Рис. 2. Робота алгоритму вимикача – закритий стан

У той же час алгоритм відмовостійкості та моніторинг даних про користувачів можуть здійснити припущення про потребу в ізоляції користувачів та ізолювати їх від інших у випадку, якщо система почне повертати помилки лише певним користувачам або групі користувачів.

Для реалізації нового підходу до балансування на основі користувачів та навантаження було вирішено переосмислити підходи шардінгу та скомбінувати їх з патерном балансувача. Зазвичай, такий алгоритм використовується у базах даних для збереження даних в різних таблицях, проте, завдяки доопрацюванню, його можна використати у сучасних додатках.

У результаті розроблено алгоритм, який ізолює потенційно небезпечного користувача або групу користувачів, що сприяє зниженню кількості потенційних помилок і повних відмов системи та надає можливість іншим користувачам ПС продовжувати використовувати додаток без перешкод.

У випадку виявлення потенційно небезпечного користувача алгоритм відмовостійкості на основі даних моніторингу здійснить розподіл навантаження користувачів за допомогою алгоритму шардінгу таким чином, щоб група підозрілих користувачів була максимально ізольована та не могла завадити використовувати програмний продукт іншим користувачам.

Схематична робота описаного алгоритму представлена на рисунку 3. В даному випадку користувача 3 ізольовано у контейнері С для того, щоб користувачі 1 та 2 могли продовжувати використовувати сервіс без перешкод.

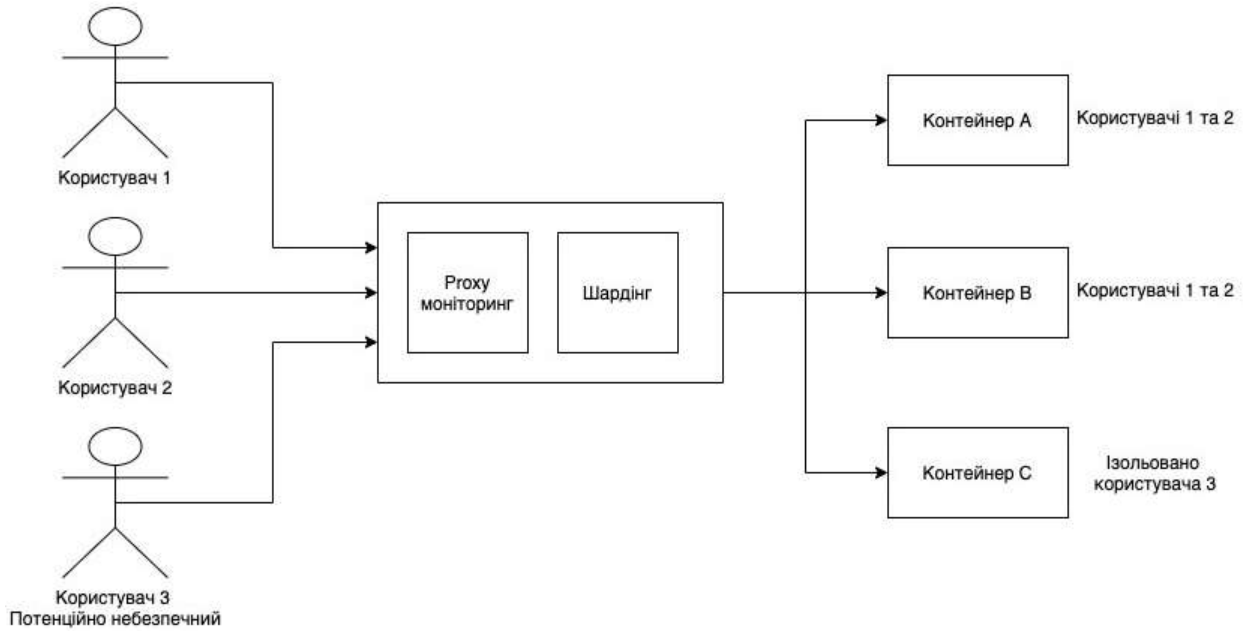

Рис. 3. Розподіл користувачів та сервісів

Висновки

Таким чином, у статті охарактеризовані основні поняття та проблеми відмовостійкості програмних систем, досліджено методи та алгоритми забезпечення відмовостійкості ПС. Зокрема, описані існуючі методи відмовостійкості ПС, на основі аналізу яких виявлені наявні проблеми у галузі відмовостійкості ПС; описані шляхи вирішення наявних проблем за допомогою реалізації нового алгоритму відмовостійкості, який дозволяє зменшити потенційні відмови ПС та проактивно реагувати на проблеми.

Література

1. Firesmith D. System resilience what exactly is it. Software Engineering Institute (SEI) at Carnegie Mellon University. URL: https://insights.sei.cmu.edu/blog/system-resilience-what-exactly-is-it/

2. Warren Axelrod C. Investing in Software resiliency. ResearchGate. URL: https://www.researchgate.net/publication/293515438_Investing_in_software_resiliency

3. Frese M. Error management or error prevention. Two strategies to deal with errors in software design. ResearchGate. URL: https://www.researchgate.net/publication/-30811276_Error_management_or_error_prevention_Two_strategies_to_deal_with_errors_in_software_design

4. Circuit breaker pattern. MICROSOFT.COM. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker

5. Retry pattern. MICROSOFT.COM. URL: https://docs.microsoft.com/en-us/azure/–architecture/patterns/retry

6. Compensation transaction pattern. MICROSOFT.COM. URL: https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction

7. Choudhury S. How data sharding works in a distributed sql database. YugabyteDB. URL: https://blog.yugabyte.com/how-data-sharding-works-in-a-distributed-sql-database/