УДК 004.9+004.774 **DOI:** 10.31673/2412-9070.2024.050183
О. В. БАРАБАШ, доктор техн. наук, професор;
ORCID: 0000-0003-1715-0761
О. В. СВИНЧУК, канд. фіз.-мат. наук, доцент;
ORCID: 0000-0001-9032-6335
Г. В. ШУКЛІН, канд. техн. наук, доцент,
ORCID: 0000-0003-2507-384X
Національний технічний університет України «Київський політехнічний університет імені Ігоря Сікорського»

АЛГОРИТМ ПЕРЕРОЗПОДІЛУ НАВАНТАЖЕННЯ ДЛЯ ЗАБЕЗПЕЧЕННЯ ФУНКЦІОНАЛЬНОЇ СТІЙКОСТІ РОЗПОДІЛЕНИХ ВЕБЗАСТОСУНКІВ

У статті розглядається алгоритм перерозподілу навантаження, який спрямований на забезпечення функціональної стійкості розподілених вебзастосунків. Функціональна стійкість є критично важливою для забезпечення надійної роботи та високої доступності вебзастосунків в умовах підвищеного навантаження, технічних несправностей або кібератак. Аналізуються різні методи перерозподілу навантаження, такі статичні та динамічні з детальним розглядом їхніх переваг та недоліків. Також проводиться аналіз існуючого програмного забезпечення, яке використовує балансувальник навантаження. На основі проведеного дослідження розроблена програмна система, яка забезпечує функціональну стійкість розподілених вебзастосунків на основі нового алгоритму перерозподілу навантаження і складається з таких модулів: сервіс імен, сервер та його інтеграція з сервісом імен, клієнт, балансувальник навантаження. Система автоматично додає та вилучає сервери, що будуть оброблювати користувацькі запити, виконує задачу балансування навантаження та розподіляє навантаження сервера, що відмовив. Постійний моніторинг стану серверів дозволяє вчасно виявляти можливі проблеми та оперативно реагувати на них шляхом визначення оптимального розподілу запитів на основі поточного стану серверів та перенаправлення запитів користувачів на активні сервери для зменшення затримок та покращення загальної продуктивності системи. Аналіз отриманих даних про навантаження дає змогу алгоритму приймати обґрунтовані рішення щодо перенаправлення запитів, що сприяє зниженню затримок. Проводилося математичне моделювання з метою визначення часу перерозподілу навантаження. Проаналізувавши результати, зроблено висновок, що із зростанням кількості відмов зростає час перерозподілу, що має експоненціальний характер.

Ключові слова: розподілені вебзастосунки, функціональна стійкість, алгоритм перерозподілу навантаження, запити, сервери.

Вступ

Інформаційні технології відіграють значну роль в житті людини, оскільки з їх допомогою більшість завдань, які виконує людина, можна автоматизувати, створюючи різноманітні вебзастосунки. Особливо це важливо для сфери медицини, фінансів, промисловості, енергетики тощо. Проте під час роботи вебзастосунків можуть виникати непередбачувані збої в їх роботі. Тому при створенні нових програмних продуктів потрібно забезпечувати одну з ключових властивостей – їх функціональну стійкість [1].

Вже існує багато методів забезпечення цієї властивості для таких систем, а саме алгоритми розподілу навантаження [2]. Вони допомагають розподіляти до системи від користувачів між декількома вузлами-обробниками, таким чином, забезпечуючи стабільну роботу вебзастосунку.

Ключовими аспектами функціональної стійкості є:

- коректність – система повинна давати точні та дійсні результати відповідно до її специфікацій і вимог;
- надійність – система має бути стабільною у своїй роботі, обробляти різні сценарії, вхідні дані та навантаження без збоїв або неочікуваних ситуацій;
- узгодженість – поведінка системи має бути узгодженою в різних випадках і за різних умов;
- відповідність – система повинна відповідати встановленим стандартам, правилам і специфікаціям, що включає сумісність із галузевими стандартами, стандартами безпеки та будь-якими іншими відповідними вказівками;
- взаємодія – у випадках, коли система взаємодіє з іншими системами або компонентами, функціональна цілісність передбачає забезпечення безперебійної та ефективної взаємодії;
- масштабованість – для систем, яким необхідно обробляти різні навантаження та вимоги користувачів, функціональна цілісність передбачає можливість ефективного масштабування без шкоди для продуктивності чи функціональності.

Функціональна цілісність має вирішальне значення для надання високоякісних ІТ-рішень. Вона часто забезпечується шляхом ретельного тестування, дотримання найкращих практик у розробці програмного забезпечення та проєктування.

Аналіз літературних даних та постановка проблеми

Розв'язанням задач забезпечення функціональної стійкості також присвячено низку наукових праць. Для забезпечення функціональної стійкості багатомодульних інформаційних систем проводять моніторинг параметрів їх технічного стану [3], визначають критерії для функціонально стійких складних систем [4]. Розроблені методи для забезпечення відмовостійкості інформаційної системи [5], методи для діагностики її несправностей [6]. У статті [7] науковцями запроповновано методологію побудови ефективної системи самодіагностики інформаційних систем на прикладі підприємств металургійної та енергетичної промисловості. Проведений аналіз свідчить, що для всіх інформаційних систем дуже важливо здійснювати моніторинг параметрів справного стану обчислювальних пристроїв, проводити їх самодіагностику, ідентифікувати та локалізовувати несправні модулі.

Проте в розглянутих роботах не розглядається питання розподілу вхідних запитів від користувача між деякою кількістю обчислювальних вузлів, що є теж важливим для надійної роботи інформаційної системи. Наразі існують різні алгоритми розподілу навантаження, а саме статичні та динамічні [8].

Алгоритми статичного розподілу навантаження розподіляють вхідне навантаження або мережний трафік між серверами на основі попередньо визначеного та фіксованого набору правил. Ці правила зазвичай встановлюються під час початкового налаштування системи та залишаються постійними. Статичне балансування навантаження не враховує зміни продуктивності сервера, умов трафіку або навантаження на систему в реальному часі [8]. Алгоритми статичного розподілу навантаження мають такі характеристики:

- правила розподілу навантаження визначаються заздалегідь і залишаються незмінними;
- забезпечують передбачуваність розподілу навантаження;
- правила розподілу навантаження є фіксованими, тому дане балансування може погано адаптуватися до динамічних змін у системі, продуктивності серверів;
- підходять для сценаріїв, коли робоче навантаження та умови системи залишаються відносно стабільними протягом тривалого часу.

Алгоритми динамічного балансування навантаження адаптуються в режимі реального часу до змін у системі. Ці алгоритми постійно відстежують такі фактори, як ефективність сервера, доступні ресурси та вхідний трафік, щоб динамічно регулювати розподіл навантаження між серверами. Динамічне балансування навантаження спрямоване на оптимізацію використання

ресурсів і забезпечення ефективної обробки різноманітних навантажень [8]. Характеристики алгоритмів динамічного балансування навантаження:

- безперервно аналізують стан системи та миттєво коригують розподіл навантаження;
- оптимізують продуктивність системи, скорочують час відповіді та запобігають перевантаженню окремих серверів;
- сприяють підвищенню відмовостійкості, перерозподіляючи робоче навантаження від серверів, які мають проблеми, що покращує надійність системи та мінімізує перебої в обробці запитів;
- часто складніші за статичні аналоги, оскільки включають постійний моніторинг і процеси прийняття рішень, що може призвести до додаткових обчислювальних витрат;
- добре підходять для середовищ із різним робочим навантаженням, непередбачуваними моделями трафіку та динамічними змінами умов системи.

На практиці вибір між статичним і динамічним балансуванням навантаження залежить від конкретних вимог системи, характеру навантаження та рівня адаптивності, необхідного для обробки змін у системі. Деякі системи можуть навіть використовувати комбінацію обох підходів, використовуючи статичне балансування навантаження для базового розподілу та динамічне балансування навантаження для обробки коливань і оптимізації в режимі реального часу.

Розглянемо існуюче програмне забезпечення, яке використовує балансувальник навантаження.

NGINX – веб-сервер із відкритим кодом і зворотний проксі-сервер, який використовується як балансувальник навантаження. NGINX допомагає розподіляти вхідний мережний трафік між декількома серверами, щоб забезпечити оптимальне використання ресурсів, підвищити продуктивність додатків і забезпечити високу доступність [9]. NGINX працює як зворотний проксі-сервер, обробляючи вхідні клієнтські запити та пересилаючи їх на один або кілька внутрішніх серверів, і може розподіляти HTTP, HTTPS, TCP і UDP трафік. Підтримує різні алгоритми балансування навантаження, зокрема Round Robin, Least Connections, IP Hash.

AWS (Amazon Web Services) надає кілька рішень для балансування навантаження, які допомагають розподіляти вхідний мережний трафік між кількома цілями, забезпечуючи високу доступність, відмовостійкість і ефективне використання ресурсів [10]. Основні функції балансувальників навантаження AWS включають: інтеграцію автоматичного масштабування, перевірки стану серверів, безпеку, балансування навантаження між зонами, маршрутизацію на основі вмісту. Вибір типу балансувальника навантаження залежить від конкретних вимог додатку та характеру трафіку, який він повинен обробляти. Пропонується кілька типів балансувальників навантаження: Classic Load, Application Load Balancer, Network Load Balancer, Gate-way Load Balancer [11].

Метою дослідження є розробка програмного забезпечення для забезпечення функціональної стійкості розподілених вебзастосунків на основі алгоритму перерозподілу навантаження.

Основна частина

Розподілені вебзастосунки відносяться до типу системної архітектури, в якій різні компоненти або програми розподіляються на кількох серверах або обчислювальних ресурсах. Ця архітектура розроблена для покращення масштабованості, продуктивності та надійності шляхом розподілу робочого навантаження та завдань обробки між різними вузлами в мережі. Дані застосунки часто використовують хмарні обчислення та мікросервісну архітектуру для досягнення своїх цілей. Хоча вони приносять численні переваги, вирішення проблем, пов'язаних із розподіленими системами, потребує ретельного проєктування, впровадження та управління.

Балансувальники навантаження (Load balancer) є критично важливими компонентами мережної архітектури, які розподіляють вхідний мережний трафік між декількома серверами, забезпечуючи оптимальне використання ресурсів, максимізуючи пропускну здатність, мінімізуючи час відповіді та запобігаючи перевантаженню будь-якого окремого сервера. Вони відіг-

рають ключову роль у покращенні продуктивності, доступності та надійності програм. Балансувальники навантаження працюють на різних рівнях моделі OSI, включаючи транспортний рівень і прикладний рівень [12]. Основні функції балансувальників навантаження:

- рівномірно розподіляють вхідний мережний трафік між кількома серверами;
- підвищують доступність і надійність програм, перенаправляючи трафік із серверів, на яких можуть виникнути проблеми;
- підтримують горизонтальну масштабованість, дозволяючи додавати нові сервери до пулу серверів;
- використовують різні алгоритми, щоб визначити, як розподілити трафік між серверами, причому вибір алгоритму залежить від таких факторів, як потужність сервера та бажана стратегія розподілу;
- перевіряють працездатність серверів, щоб переконатися, що вони активні, і якщо сервер виявлено як несправний, балансувальник навантаження припиняє надсилати трафік на цей сервер, доки він не відновиться;
- підтримують постійність сесії, гарантуючи, що запити від одного клієнта постійно спрямовуються на той самий сервер.

Балансувальники навантаження мають важливе значення для вирішення складних завдань сучасних вебзастосунків, забезпечення ефективного використання ресурсів, підвищення продуктивності та забезпечення зручності для користувачів. Вони є фундаментальним компонентом у створенні масштабованих, стійких і високопродуктивних розподілених систем.

При роботі з балансувальником навантаження деякі сервери з різних причин можуть стати недоступними. Тому постає проблема в обробці запитів, які оброблялися цим сервером в той момент, коли стало відомо, що він перестав працювати. Для цього будемо використовувати інші активні сервери. Механізм повторних спроб націлений на тимчасові помилки, такі як перевищення часу очікування підключення або помилки на стороні сервера. Щоб уникнути перевантаження сервера, який має проблеми, балансувальник повинен використовувати експоненціальне збільшення часу очікування для повторної відправки запиту. Також для балансувальника навантаження можна налаштувати політику повторних спроб, які визначатимуть такі параметри, як максимальна кількість повторних спроб, максимальна тривалість повторних спроб і умови, за яких від повторних спроб слід відмовлятися. Балансувальники навантаження можуть реалізувати повторні спроби на рівні HTTP. Наприклад, якщо внутрішній сервер повертає код статусу 5xx, балансувальник навантаження може автоматично повторити запит.

Отже, для запобігання таких проблем необхідно встановити відповідний максимальний час очікування для повторних спроб, щоб уникнути тривалих затримок. Також необхідно встановити обмеження на кількість повторних спроб, щоб запобігти появі нескінченного циклу.

Розроблена програмна система, що забезпечує функціональну стійкість розподілених вебзастосунків, складається з модулів:

- сервіс імен;
- сервер та його інтеграція з сервісом імен;
- клієнт;
- балансувальник навантаження.

Сервіс імен – це модуль, що реєструє обробників користувацьких запитів та сервери. При реєстрації сервер надає запит до сервісу імен та повідомляє свої дані. Сервіс імен постійно перевіряє стан кожного із зареєстрованих серверів, і якщо якийсь із них не відповідає, то цей сервер буде видалено із реєстру.

Сервер – це модуль, що обробляє користувацькі запити. Він може бути представленим у декількох екземплярах і тому потребує балансування для забезпечення ефективної обробки запитів.

Клієнт – поєднує в собі як функції зовнішнього інтерфейсу, так і функції балансувальника. Балансувальник навантаження є зворотним проксі. Він є віртуальною IP-адресою (VIP), що представляє програму для клієнта. Клієнт підключається до VIP, а балансу-вальник наванта-

ження приймає рішення за допомогою своїх алгоритмів, щоб надіслати з'єднання до конкретного екземпляра програми на сервері. Балансувальник навантаження продовжує керувати та контролювати з'єднання протягом усього часу.

На рис. 1 представлений алгоритм перерозподілу навантаження в умовах нормальної роботи системи. Після запуску програмних модулів сервер з'єднується з сервісом імен, щоб зареєструватися в ньому. Після сервіс імен починає постійно перевіряти стан сервісу через health check. Коли користувач надсилає запит, він потрапляє в балансувальник навантаження. Балансувальник отримує інформацію від сервісу імен про доступні сервери і за допомогою алгоритму перерозподілу навантаження визначає, якому сервісу буде перенаправлено цей запит. Запит потрапляє до сервера і після обробки сервер повертає відповідь балансувальнику, а він в свою чергу клієнту.

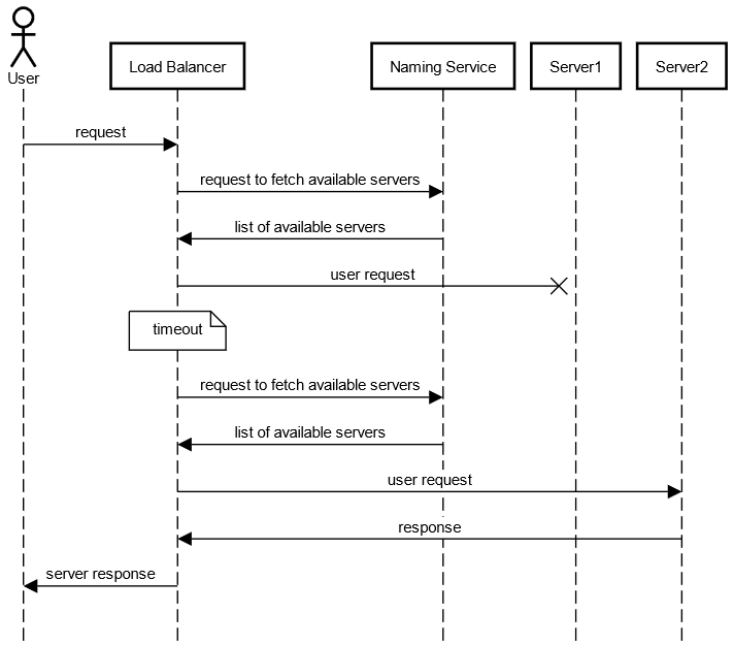

Рис. 1. Алгоритм перерозподілу навантаження при нормальній роботі системи

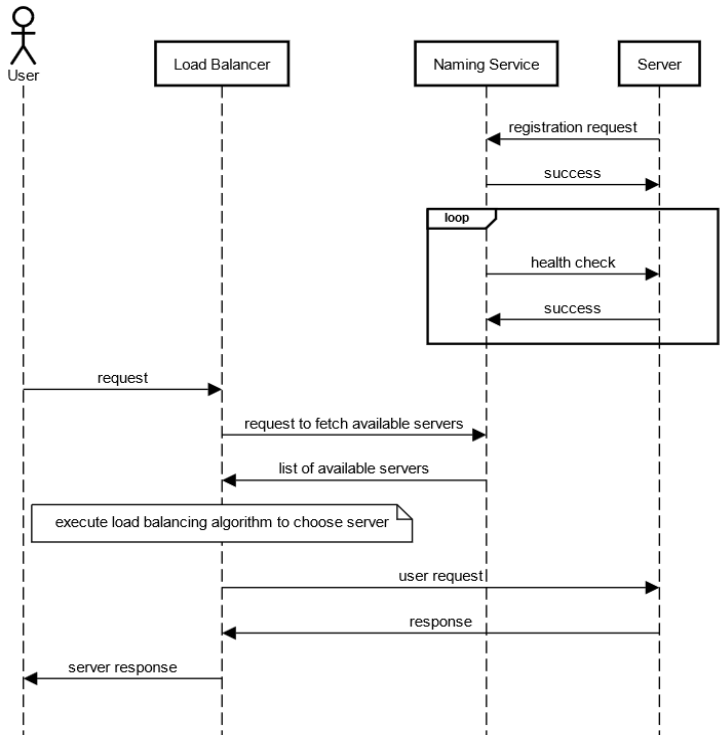

Рис. 2. Алгоритм перерозподілу навантаження системи у випадку відмови сервера

Розглянемо алгоритм перерозподілу навантаження для випадку відмови одного із серверів (рис. 2). Спочатку повторюється процедура реєстрації серверів у сервісі імен. Користувач надсилає запит, балансувальник отримує список доступних серверів і за допомогою алгоритму розподілу навантаження визначає сервер, що буде обробляти запит. Після того як сервер отримав запит від балансувальника, в ньому сталася неочікувана помилка і він став недоступним. У цей час сервіс імен не отримав позитивної відповіді на health check від недоступного сервера і вилучив його зі списку активних серверів. Після цього пройшов час очікування на відповідь від сервера у балансувальника і він не отримавши відповіді й намагається обробити запит повторно. Від сервісу імен він отримує оновлений список серверів, в якому відсутній неактивний сервер і після повторного застосування алгоритму перерозподілу навантаження запит відправляється активному серверу. Сервер, обробивши запис, відправляє його балансувальнику, а він – клієнту.

Для того, щоб щоб запустити програмну систему, у користувача повинна бути встановлена віртуальна машина Java (JVM) 17. Спочатку необхідно налаштувати систему. Спочатку необхідно налаштувати систему. У конфігураційних файлах серверів прописати адресу сервісу імен, щоб сервери могли в ньому зареєструватися, а у клієнті прописати адресу сервісу імен, щоб балансуваль-

ник навантаження мав можли-вість отримувати від нього список доступних серверів.

Після налаштування за допомогою команди java -jar <назва файлу> користувач запускає сервіс імен, потім за допомогою тієї ж команди запускає декілька серверів і клієнт. Коли всі компоненти запустяться, користувач може за допомогою веб-браузера перейти на адресу, що оброблюється серверами, та спробувати перевірити роботу системи.

Після успішного запуску всіх компонентів системи всі сервери автоматично повинні додатися до реєстру додатків у сервісі імен, також клієнт повинен встановити зв'язок із сервісом імен для отримання списку доступних серверів. Сервіс імен повідомляє про реєстрацію додатків у логах.

Під час кожного запиту клієнт отримує від сервісу імен список доступних серверів та за допомогою алгоритмів перерозподілу навантаження обирає оптимальний сервер для обробки запиту. Коли сервер отримує запит, він повідомляє про це у своїх логах (рис. 3).

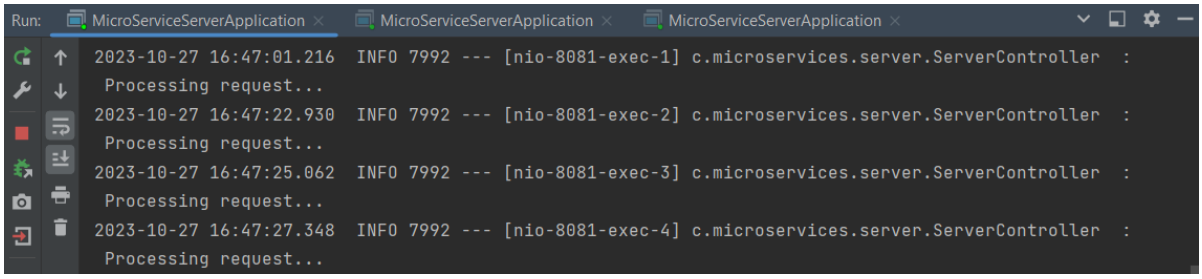

Рис. 3. Повідомлення у логах сервера про обробку запитів

У логах клієнта також можна побачити, який сервер обробляв запит. Щоб перевірити чи вилучається непрацюючий сервіс із реєстру в сервісі імен, достатньо вимкнути будь-який із сервісів. У логах сервісу імен повинно з'явитися повідомлення про вилучення недоступного сервера. Після вимкнення серверів всі запити повинні оброблятися єдиним активним сервером.

Таким чином, система автоматично додає та вилучає сервери, що будуть оброблювати користувацькі запити, система виконує задачу балансування навантаження та розподіляє навантаження сервера, що відмовив.

Для підтвердження правильності роботи алгоритмів проведено математичне моделювання процесу перерозподілу навантаження вебзастосунку, який може містити різну кількість серверів N, наприклад, 15, 20. Дане обмеження обумовлено обмеженістю обчислювальної потужності сучасної мікропроцесорної техніки під час рішення класу NP-повних задач. Моделювання проводилося з метою визначення часу перерозподілу навантаження tr – часу відмови деякого серверу до моменту перерозподілу його задач на інші активні сервери, визначається в умовних одиницях часу (уоч). Передбачається, що після успішного запуску всіх компонентів системи сервери мають певну апріорну ймовірність працездатного стану, яка позначається p та варіюється під час моделювання у таких значеннях: 0,8; 0,85; 0,9; 0,95. Також задається заданий рівень достовірності $Dзад$ моніторингу серверів, який виконує роль ознаки припинення накопичування результатів перевірок в пам'яті, що буде виконувати алгоритм перерозподілу навантаженння. Задана достовірність варіюється із такими значеннями: 0,8; 0,85; 0,9; 0,95; 0,98. За результатами моделювання побудовані графіки залежності часу перерозподілу навантаження tr від числа серверів $Nвід$, що відмовили для різних значень достовірності $Dзад$ та апріорної ймовірності працездатності серверів p=0,95, що обробляють запити користувачів (рис. 4-5).

Проаналізувавши графіки, можна зробити висновок, що із зростанням кількості відмов зростає час пеперозподілу. Дане зростання має експоненціальний характер, що обумовлено особливостями технології накопичення результатів моніторингу. При значній кількості відмов $Nвід>N$ є необхідність накопичення значної кількості результатів перевірок для того, щоб підмножина активних серверів перевірила всі сервери, що входять до множини неактивних з яко-

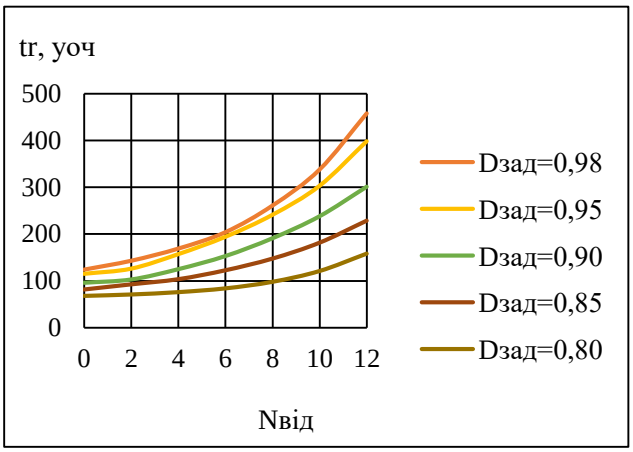

Рис. 4. Залежність часу перерозподілу навантаження *tr* від числа серверів *Nвід*, що відмовили, за умови *Dзад*, *p*=0,95, *N*=15

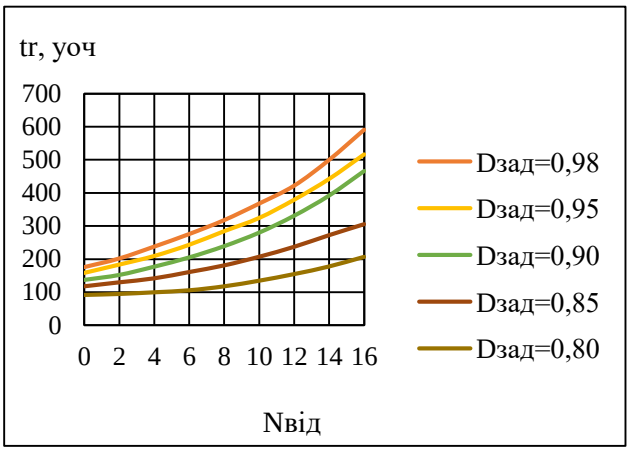

Рис. 5. Залежність часу перерозподілу навантаження *tr* від числа серверів *Nвід*, що відмовили, за умови *Dзад*, *p*=0,95, *N*=20

мога більшою кількістю перевірок. У такому випадку є можливість видачі результату із заданою достовірністю.

Отже, процедура перерозподілу навантаження є збіжною. Для будь-якої відмовної ситуації, що задавалась випадковим чином, запити з неактивних серверів були перерозподілені на активні і це було зроблено за обмежений час.

Висновки

Функціональна стійкість вебзастосунків є ключовим аспектом забезпечення їх надійної роботи та високої доступності для користувачів. Вона включає в себе здатність системи підтримувати безперервну роботу під час збоїв, атак або надмірного навантаження. Одним із основних методів підвищення функціональної стійкості є використання алгоритму перерозподілу навантаження між серверами, коли система працює нормально та у випадку, коли відмовляють сервери.

Алгоритм перерозподілу навантаження сприяє рівномірному розподілу запитів від клієнтів між кількома серверами, що забезпечує оптимальне використання ресурсів та мінімізує ризик перевантаження окремих серверів. Основними функціями цього алгоритму є постійний моніторинг за станом серверів та рівнем навантаження на кожному з них, визначення оптимального розподілу запитів на основі поточного стану серверів та перенаправлення запитів користувачів на активні сервери для зменшення затримок та покращення загальної продуктивності системи.

Для підтвердження правильності роботи алгоритму проведено математичне моделювання процесу перерозподілу навантаження вебзастосунку, який може містити різну кількість серверів. Моделювання проводилося з метою визначення часу перерозподілу навантаження *tr*, тобто часу відмови деякого серверу до моменту перерозподілу його задач на інші активні сервери, визначається в умовних одиницях часу. Проаналізувавши результати, зроблено висновок, що із збільшенням кількості відмов зростає час пеперозподілу. Дане зростання має експоненціальний характер.

Отже, застосування алгоритму перерозподілу навантаження дозволяє забезпечити високу доступність та функціональну стійкість вебзастосунків, ефективне використання серверних ресурсів та покращену користувацьку взаємодію.

Список літератури

1. Собчук В.В., Барабаш О.В., Мусієнко А.П. Основи забезпечення функціональної стійкості інформаційних систем підприємств в умовах впливу дестабілізуючих факторів: монографія. Київ: Міленіум, 2022. 272 с. ISBN: 973-966-8063-82-3. URL:

https://www.researchgate.net/publication/363474851_Basis_for_functional_stability_of_information_systems_businesses_under_the_influence_of_destabilizing_factors

2. Shafiq D.A., Jhanjhi N., Abdullah A. Load balancing techniques in cloud computing environment: a review. Journal of King Saud University - Computer and Information Sciences. 2022. Vol. 34, no. 7. P. 3910-3933. URL: https://doi.org/10.1016/j.jksuci.2021.02.007

3. Barabash O., Svynchuk O., Salanda I., Mashkov V., Myroniuk M. Ensuring the Functional Stability of the Information System of the Power Plant on the Basis of Monitoring the Parameters of the Working Condition of Computer Devices. Advanced Information Systems, 2024. Vol. 8, no. 2. P. 107-117. URL: https://doi.org/10.20998/2522-9052.2024.2.12

4. Barabash, O., Sobchuk, V., Musienko, A., Laptiev, O., Bohomia, V., Kopytko, S. System Analysis and Method of Ensuring Functional Sustainability of the Information System of a Critical Infrastructure Object. In: Zgurovsky, M., Pankratova, N. (eds) System Analysis and Artificial Intelligence. Studies in Computational Intelligence, 2023. Vol. 1107. Springer, Cham. P. 117-192. URL: https://doi.org/10.1007/978-3-031-37450-0_11

5. Peng S.-L., Lin C.-K., Tan J.J.M. and Hsu L.-H. The g-Good-Neighbor Conditional Diagnosability of Hypercube under PMC Model. Applied Mathematics and Computation, 2012. Vol. 218, no. 21. P. 10406-10412. URL: https://doi.org/10.1016/j.amc.2012.03.092

6. Yuan J., Liu A., Ma X., Liu X., Qin X. and Zhang J. The g-Good-Neighbor Conditional Diagnosability of k-Ary n-Cubes under the PMC Model and MM Model. IEEE Transactions on Parallel and Distributed Systems, 2015. Vol. 26, no. 4. P. 1165-1177. URL: http://doi.org/10.1109/TPDS.2014.2318305

7. Sobchuk V., Barabash O., Musienko A., Svynchuk O. Adaptive accumulation and diagnostic information systems of enterprises in energy and industry sectors. E3S Web of Conferences. 2021. Vol. 250. P. 82-87. URL: https://doi.org/10.1051/e3sconf/202125008002

8. Сисоєв І.К., Гавриленко В.В. Адаптивний алгоритм балансування навантаження в додатках з використанням технології контейнеризації. Системи управління, навігації та зв'язку. Збірник наукових праць. 2022. Т. 1 (67). С. 81-83. URL: https://doi.org/10.26906/SUNZ.2022.1.081

9. HTTP Load Balancing | NGINX Documentation. URL: https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/ (дата звернення: 27.06.2024).

10. Service load balancing – Amazon Elastic Container Service. URL: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-load-balancing.html (дата звернення: 13.06.2024).

11. Feig R. Computer Networks: The OSI Reference Model. URL: http://www.rad.com/networks/1994/osi/intro.htm (дата звернення: 27.06.2024).

12. Zaouch A., Benabbou F. Load Balancing for Improved Quality of Service in the Cloud. International Journal of Advanced Computer Science and Applications. 2015. Vol. 6, no. 7. P. 184-189. URL: https://doi.org/10.14569/IJACSA.2015.060724

O. Barabash, O. Svynchuk, H. Shuklin

LOAD REDISTRIBUTION ALGORITHM FOR ENSURING FUNCTIONAL STABILITY OF DISTRIBUTED WEB RELATIONS

The article considers the load redistribution algorithm, which is aimed at ensuring the functional stability of distributed web applications. Functional stability is critical to ensure reliable operation and high availability of web applications in conditions of increased load, technical malfunctions or cyber attacks. Different methods of load redistribution, both static and dynamic, are analyzed with a detailed consideration of their advantages and disadvantages. There is also an analysis of existing software that uses a load balancer. The developed software system, which ensures functional stability of distributed web applications, consists of modules: name service, server and its integration with the name service, client, load balancer. The load redistribution algorithm is pre-sented in the conditions of normal system operation and in case of server failure. Constant monitoring of server status allows timely detection of possible problems and prompt response to them by determining the optimal distri-

bution of requests based on the current state of servers and redi-recting user requests to active servers to reduce delays and improve overall system performance. Analysis of the received load data allows the algorithm to make informed decisions about the redirection of requests, which helps to reduce delays.

Mathematical modeling was carried out in order to determine the time of load redistribution, that is, the time of failure of some server until the moment of redistribution of its tasks to other active servers, determined in conventional time units. After analyzing the results, it was concluded that with an increase in the number of failures, the time of paper distribution increases, which has an exponential character. Therefore, the application of the load redistribution algorithm allows to ensure high availability and functional stability of web applications, efficient use of server resources and improved user interaction.

Keywords: distributed web applications, functional stability, load redistribution algorithm, requests, servers.