УДК 004.415.2
DOI https://doi.org/10.32782/2663-5941/2025.2.2/06

Вовченко Д. С.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Олещенко Л. М.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

АНАЛІЗ АЛГОРИТМІВ БАЛАНСУВАННЯ ВУЗЛІВ В ПРОГРАМНИХ СИСТЕМАХ РОЗПОДІЛЕНОЇ ОБРОБКИ ВЕЛИКИХ ДАНИХ

Алгоритми балансування навантаження відіграють важливу роль у розподілених системах обробки великих даних, забезпечуючи ефективне використання ресурсів, зменшення часу виконання запитів та підвищення надійності роботи. Вибір конкретного алгоритму залежить від характеристик системи, типу навантаження та вимог до продуктивності. У статті представлені актуальні алгоритмічні рішення для вирішення проблеми розподілення навантаження у системах з багатьма вузлами. Проаналізовано існуючі алгоритми балансування вузлів, зокрема, Least Loaded, Fair Scheduling, DAG Scheduler, Dynamic Resource Allocation, Least Request, Weighted Round Robin, Consistent Hashing, Adaptive Load Balancing та Partition-Aware Load Balancing, які спрямовані на покращення загальної ефективності розподілених систем обробки великих даних.

Метою даної статті є дослідження ефективності сучасних рішень, висвітлення проблем часу відповіді серверу, обробки запитів, використання ресурсів, а також обмеження при дослідженні великих даних. Аналіз алгоритмів балансування навантаження вузлів у розподілених системах обробки великих даних показав, що використання таких методів, як Dynamic Resource Allocation у Apache Spark, дозволяє підвищити пропускну здатність на 30–50 %, Adaptive Load Balancing у Netflix зменшує час очікування запитів до 40 % у пікові періоди, а Consistent Hashing у Google Cloud Load Balancer мінімізує перенаправлення запитів при зміні конфігурації кластера, що підвищує стабільність системи; проте основними недоліками розглянутих алгоритмів є обмежена адаптивність до непередбачуваних змін у навантаженні, затримки при масштабуванні, а також складність налаштування та обчислювальні витрати на динамічний перерозподіл ресурсів.

Подальші дослідження спрямовані на розробку алгоритму балансування вузлів з урахуванням усіх переваг та недоліків сучасних рішень. Запропонований удосконалений алгоритм балансування вузлів дозволить уникнути проблеми перевантаження одного з вузлів завдяки використанню технологій динамічного розрахунку ваги вузлів та машинного навчання на прикладах роботи в сучасних системах з урахуванням параметрів ефективності при обробці великих даних.

Ключові слова: *великі дані, розподілені програмні системи з багатьма вузлами, алгоритми балансування вузлів, CPU, серверна пам'ять, робота з базами даних, машинне навчання.*

Постановка проблеми. Обробка великих даних потребує ефективного розподілу обчислювальних ресурсів для забезпечення високої продуктивності, надійності та мінімізації часу виконання запитів. Одним із ключових аспектів розподілених систем є балансування навантаження між вузлами кластера. Ефективні алгоритми балансування навантаження забезпечують оптимальне використання обчислювальних потужностей, запобігають перевантаженню окремих серверів і підвищують загальну продуктивність системи. Усе більше сучасних вебзастосунків стикається з проблемою обробки великих даних – наборів інформації надвеликих розмірів, оскільки це потребує великої кількості ресурсів. Одним із шляхів її вирішення є розбиття серверу на декілька вузлів для досягнення масштабованості системи. Додавання нового вузла фактично створює новий сервер з власними ресурсами, який можна використати для кращої обробки інформації. Одним із найбільш поширених недоліків цього підходу є проблема з коректним розподіленням навантаження між вузлами. Доволі поширена ситуація – коли один з вузлів перенавантажений або, навпаки, недовантажений, що призводить до зниження швидкодії подібних систем шляхом створення черг вхідних запитів на одному з вузлів і збільшення загального часу їх

обробки. Щоб уникнути цієї ситуації, було створено ряд алгоритмів для балансування навантаження між вузлами [1]. Аналізуючи такі параметри, як пропускна здатність, кількість оброблених запитів, середній час обробки, вони розподіляють вхідний потік інформації на той чи інший вузол. Наразі запропоновано багато алгоритмів, деякі змінюють параметри для аналізу, експериментують з іншими алгоритмами, або ж використовують такі технології, як машинне навчання. Кожен з запропонованих варіантів модифікації має свої переваги та недоліки, однак, багато цих досліджень мало приділяють увагу проблемі обробки великих даних, що є особливо актуально при динамічному їх розподілені у вже працюючих системах [2]. При появі подібних проблем часто спостерігається зниження загальної продуктивності програмних систем через перенавантаження одного з вузлів.

Сучасні системи розподіленої обробки великих даних активно використовують алгоритми балансування навантаження для забезпечення рівномірного розподілу обчислювальних ресурсів між вузлами. Проте, незважаючи на значний прогрес у цій галузі, існує низка невирішених проблем, які обмежують ефективність таких систем.

В умовах змінного потоку даних традиційні алгоритми балансування, такі як Round Robin або статичний розподіл ваг, можуть демонструвати низьку ефективність. Вузли можуть бути перевантажені або недовантажені через нерівномірний розподіл задач.

У розподілених системах вузли можуть мати різну продуктивність, обсяг пам'яті та мережеву пропускну здатність. Багато класичних алгоритмів не враховують ці фактори, що призводить до неефективного використання ресурсів. Передача великих обсягів даних між вузлами може створювати додаткові затримки, особливо в умовах обмеженої мережевої пропускної здатності. Відсутність ефективних алгоритмів, що враховують ці обмеження, знижує загальну продуктивність системи.

Багато систем балансування не враховують можливі збої вузлів або варіації продуктивності в реальному часі. Відсутність механізмів адаптації до змін у доступності ресурсів може спричинити втрату продуктивності або навіть збої в обробці даних.

Сучасні алгоритми балансування рідко оптимізовані для мінімізації енергоспоживання, що стає критично важливим у масштабних обчислювальних кластерах та хмарних середовищах.

Зважаючи на вищезазначені проблеми, виникає необхідність у розробці удосконаленого алгоритму балансування вузлів у системах розподіленої обробки великих даних, який враховуватиме зміну навантаження у реальному часі та адаптуватиметься до поточного стану ресурсів; враховуватиме гетерогенність обчислювальних вузлів, оптимізуючи їх використання; мінімізуватиме затримки передачі даних шляхом ефективного розподілу задач між вузлами; підвищуватиме відмовостійкість системи за рахунок автоматичного перерозподілу навантаження у разі виходу вузлів з ладу; забезпечуватиме оптимізацію енергоспоживання для зниження експлуатаційних витрат.

Отже, актуальним є розробка нового підходу до балансування вузлів, що використовуватиме методи машинного навчання та динамічне обчислення ваги вузлів на основі їхньої поточної продуктивності та навантаження. Це дозволить підвищити ефективність розподілених систем обробки великих даних, забезпечуючи швидкість, надійність та оптимальне використання ресурсів.

Аналіз останніх досліджень і публікацій. Алгоритми балансування навантаження вузлів поділяються на статичні та динамічні.

Статичні методи передбачають розподіл навантаження на початковому етапі роботи системи без можливості адаптації до змінних умов виконання. Розглянемо основні завдання алгоритмів цього типу.

Round Robin – використовує рівномірний розподіл запитів між серверами по черзі.

Least Connections – використовує спрямування запитів на сервери з найменшою кількістю активних з'єднань.

Weighted Round Robin – використовує розподіл запитів з урахуванням продуктивності вузлів.

Динамічні методи адаптуються до поточного стану системи, забезпечуючи рівномірне навантаження в реальному часі. До основних алгоритмів цього типу належать наступні.

Least Response Time – використовує спрямування запитів на сервери з найнижчим часом відгуку.

Adaptive Load Balancing – використовує штучний інтелект та машинне навчання для оптимізації розподілу навантаження.

Work Stealing – використовує динамічне перерозподілення завдань між вузлами в разі їх нерівномірного завантаження.

Сучасні програмні системи для розподіленої обробки великих даних, такі як Apache Hadoop, Apache Spark та Kubernetes, використовують різні

алгоритми балансування навантаження, зокрема, Hadoop YARN розподіляє ресурси на основі вимог до пам'яті та процесорного часу, Apache Spark – застосовує динамічний розподіл завдань між виконавцями, Kubernetes – використовує алгоритми розподілу подів між вузлами на основі навантаження та потужності обчислювальних ресурсів.

Одним із представлених у наукових роботах оптимізованих алгоритмів балансування вузлів є покращений алгоритм на основі обрахунку ваги та мінімальних з'єднань. Базові варіанти цих підходів широко використовуються у системах з декількома вузлами і є доволі поширеними, однак, вони мають недолік. Для коректного розподілення вхідного потоку необхідно динамічно розраховувати вагу кожного з вузлів і на основі цієї ваги керувати потоком, що не є можливо в базових варіантах цих методів. Покращений алгоритм усуває цей недолік та дозволяє динамічно розраховувати вагу. Для цього алгоритм використовує кілька параметрів, зокрема, L – навантаження на сервер, C – використання CPU та M – рівень заповненості пам'яті. Крім того, також використовуються: P – кількість з'єднань, B – пропускна здатність та D – використання диску. Для всіх цих параметрів характерне наступне: $C, M, P, B, D \in [1, 0]$. Ці параметри використовуються для розрахунку навантаження на сервер:

$$L = 1 - (1 - \lambda_c C)(1 - \lambda_m M) \times$$
$$\times (1 - \lambda_p P)(1 - \lambda_b B)(1 - \lambda_d D), \quad (1)$$
$$\lambda_i \geq 0, \quad 0 \leq (1 - \lambda_r X_i) \leq 1. \quad (2)$$

Розраховуючи навантаження на вузлах таким чином, можна динамічно спостерігати за навантаженням та розподіляти вхідний потік відповідно до його значення. Для цього розраховується порогове значення ∂:

$$\partial = \left| 1 - \frac{\frac{C(S_i)}{W(S_i)}}{\left(\frac{1}{n}\right)\sum_{1}^{n}\frac{C(S_i)}{W(S_i)}} \right|. \quad (3)$$

Параметр $W(S_i)$ репрезентує поточну вагу серверу i, а $C(S_i)$ – кількість підключень до нього. З формули (1), якщо значення $L(S_i)$ менше порогового значення δ, то навантаження на сервер i вважається низьким і, навпаки, при $L(S_i) \geq \delta$ – високим. Виходячи з цього, нову динамічну формулу ваги можна представити наступним чином:

$$W(S_i) = \begin{cases} W(S_i) + \partial, & L(S_i) \leq \delta \\ W(S_i) - \partial, & L(S_i) > \delta \end{cases} \quad (4)$$

Балансування навантаження між серверами за даним алгоритмом представляється блок-схемою на Рис. 1. Представлена блок-схема описує алгоритм балансування навантаження в розподіленій системі, який динамічно перерозподіляє ресурси між серверами для забезпечення оптимальної роботи. Алгоритм починається з перевірки, чи навантаження на кожен сервер пропорційне його обчислювальним можливостям. Якщо це так, сервер продовжує надавати послуги без змін. Якщо навантаження розподілене нерівномірно, перевіряється, чи всі сервери перевищують допустиме навантаження. Якщо ж ні, система працює в поточному режимі. Якщо ж перевищення є, алгоритм коригує вагові коефіцієнти серверів, що впливають на їхній пріоритет при розподілі навантаження. Після цього вибирається сервер, який найбільше підходить для обробки нових запитів, і перевіряється, чи його поточне навантаження не перевищує допустимого рівня. Якщо навантаження в межах норми, сервер продовжує працювати. Якщо ж він перевантажений, додається новий сервер для обробки частини запитів. Такий підхід дозволяє автоматично адаптуватися до змін у навантаженні та ефективно використовувати ресурси розподіленої системи.

Якщо аналізувати показники, то даний алгоритм порівнювався зі зваженими алгоритмами Round-Robin та найменших з'єднань, і в результаті показав покращення у часі відповіді, загальному навантажені на сервері та зменшив час затримки. Однак, дане дослідження має недолік, оскільки воно було виконане лише на тестовому середовищі, що не дає повної ясності при роботі з реальними даними, особливо з великими даними.

Іншим прикладом алгоритму балансування можна навести алгоритм на основі відстані клієнта до сервера [4]. За замовчуванням, будь-яку систему з декількома вузлами можна представити наступною архітектурою (Рис. 2).

Разом усі вузли системи працюють як одне ціле, однак, кожен вузол може знаходитись на різній відстані до певного клієнта. Кількість підключень, які необхідно виконати або отримати доступ до певного вузла, можна позначити параметром h. Інший параметр, який в цьому контексті вартий уваги це – це час затримки (Round-Trip Time або RTT), який позначають як t. Він означає час, за який користувач отримує відповідь від серверу на

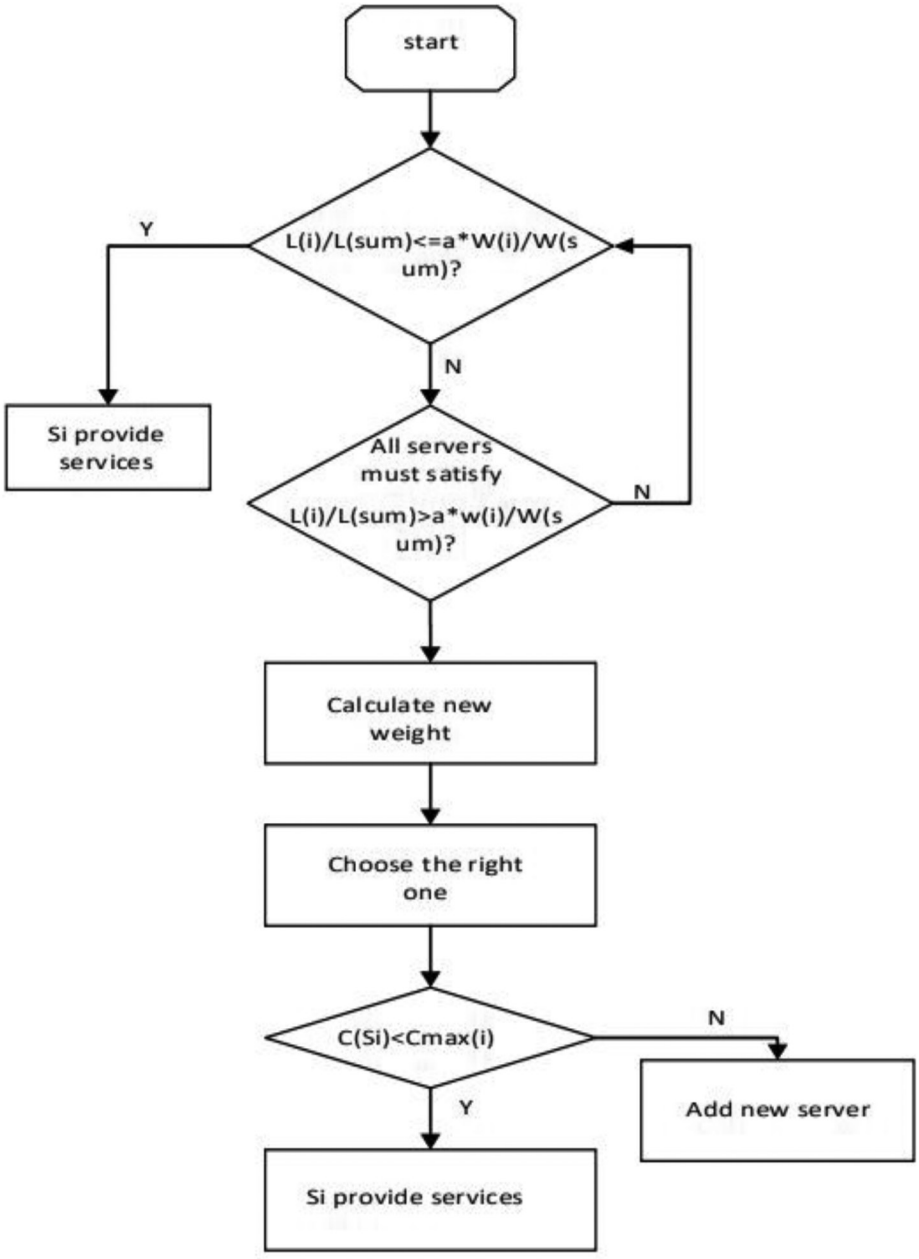

Рис. 1. Блок-схема процесу балансування навантаження між серверами [3]

свій початковий запит. На основі цих двох параметрів можна виділити значення P, що відображає відстань до серверу у даному алгоритмі:

$$P = h + t; \tag{5}$$

$$Q = \frac{1}{P}. \tag{6}$$

Значення Q репрезентує коефіцієнт вузла у алгоритмі відстані. З цього логічно випливає, що чим більше значення P, тим менше буде Q для цього вузла, дасть меншу вірогідність потрапляння запиту на нього. Чим менше значення P, тим з більшою вірогідністю цей вузол обробить запит.

Для визначення відстані до кожного з вузлів спочатку клієнт надсилає запит до всіх доступних вузлів і з отриманих відповідей розраховує значення P для них. На основі цих даних будується упорядкований масив вузлів, де на першому місці буде вузол з найменшим значення P. При подальшому надходженні наступних N запитів вони по одному розподіляються між серверами у порядку створеного масиву. Для наступних масивів запитів $N + i$ в алгоритмі вже враховується поточне навантаження на вузлах і на його основі формується новий упорядкований масив вузлів. На основі цих кроків досягається баланс між вузлами при використанні алгоритму на основі відстані

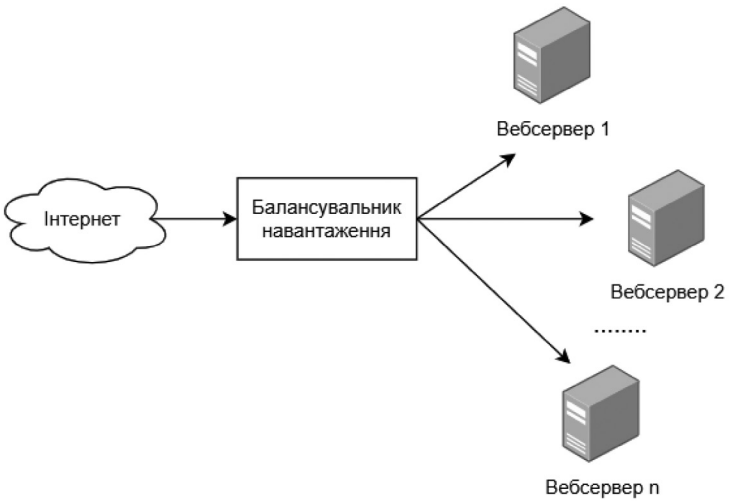

Рис. 2. Архітектура системи з декількома вузлами

між клієнтом і сервером. Даний алгоритм показав свою ефективність у порівнянні з RR алгоритмом та покращив значення часу відповіді серверу та загального навантаження.

Наступний запропонований алгоритм базується на параметрах навантаження на відповідному вузлі і використовує чотири аргументи: використання CPU, пам'яті, мережі та пропорційне відношення використаної пам'яті на контейнері [5]. Використання CPU розраховується як відношення суми витраченого CPU в користувацькому режимі та режимі центрального процесора до суми CPU у всіх трьох можливих режимах: користувацькому, режимі процесора та режимі очікування:

$$CPU_{Ratio_i(t)} = \frac{CPU_{usr(t)} + CPU_{kernel(t)}}{CPU_{usr(t)} + CPU_{kernel(t)} + CPU_{free(t)}}. \quad (7)$$

Наступний параметр використання пам'яті розраховується через відношення вільної пам'яті до загальної:

$$MEM_{Ratio_i(t)} = 1 - \frac{MEM_{free(t)}}{MEM_{total(t)}}. \quad (8)$$

Використання мережі знаходиться через відношення суми надісланих та отриманих байт до загальної пропускної здатності вузла за часовий проміжок t:

$$NET_{Ratio_i(t)} = \frac{NET_{sent(t)} + NET_{rev(t)}}{NET_{band}}. \quad (9)$$

Останній параметр відповідає за відношення використаної пам'яті на контейнері. Варто зазначити, що під контейнерами маються на увазі вузли у середовищі утиліти Docker. Ця утиліта використовується для поєднання усіх наявних вузлів системи і без неї вони не можуть працювати злагоджено. Цей параметр розраховується за формулою:

$$D_{i(t)} = \frac{\sum mem_{usage}}{\sum mem_{limit}}. \quad (10)$$

Сам алгоритм передбачає розрахунок ваги для кожного з доступних вузлів на основі чотирьох раніше зазначених параметрів. Для кожного вузла з самого початку проставляється значення ваги рівне 100, однак, від нього віднімається значення параметру навантаження в реальному часі $L_i(t)$, формула якого:

$$L_i(t) = \alpha_1 \cdot CPU_{Ratio_i(t)} + \alpha_2 \cdot MEM_{Ratio_i(t)} + \\ + \alpha_3 \cdot NET_{Ratio_i(t)} + \alpha_4 \cdot D_{i(t)}. \quad (11)$$

Коефіцієнти α обираються довільно, однак вони мають відповідати наступним вимогам:

$$\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1 \quad 0 < \alpha_i < 1. \quad (12)$$

Виходячи з цього, значення ваги на поточному вузлі розраховується як:

$$W_i(t) = 100 - L_i(t). \quad (13)$$

Варто зазначити, що значення ваги – це динамічний параметр, який розраховується в кожний заданий момент часу t та може змінюватись для кожного вузла. Іншими словами, для кожного вузла в окремий момент часу є своє значення $W_i(t)$ і в залежності від нього система обирає, на який з вузлів краще перенаправити вхідні запити.

В результаті тестування даний алгоритм позитивно вплинув на швидкість відповіді серверу у порівнянні з базовими RR та алгоритмом на основі найменших з'єднань.

Варто зазначити, що в даній роботі не розглядаються статичні алгоритми балансування вузлів, вони обчислюються вагу кожного вузла на основі його конфігурації і в процесі роботи вона не змінюється. Це може призвести до перевантаження одного з вузлів і в той же час, недовантаження іншого. Динамічні алгоритми набагато практичніші за рахунок можливості змінювати вагу через певні проміжки часу.

Сучасні динамічні алгоритми часто відрізняються лише тими параметрами, на основі яких розраховується вага і як приклад цього можна навести інший алгоритм [6], який використовує частоту з'єднань до вузла та навантаження на буферну зону ядра, також відому як Kernel Socket Buffer.

Частота з'єднань до вузла розраховується як відношення кількості з'єднань T на певному вузлі системи за проміжок часу до середньої кількості з'єднань усієї системі, де N – кількість вузлів системи:

$$L(T)_i = T_i \cdot \left(\frac{\sum_{i=1}^{N} T_i}{N} \right)^{-1} . \qquad (14)$$

У системі Linux пакети проходять через протоколи TCP/IP та загальне програмне забезпечення мережевого драйвера. Все це відбувається завдяки буферу сокетів Kernel Socket Buffer, який відповідає за передачу повідомлень між джерелом і приймачем. Крім цих двох параметрів алгоритм також використовує стандартні для багатьох подібних алгоритмів значення утилізації CPU та пам'яті на вузлі. З певною періодичністю часу система збирає інформацію про навантаження, CPU, частоту з'єднань та буферу сокетів з кожного з вузлів. У разі відсутності зв'язку з певним вузлом, його вага змінюється на 0 і він більше не буде отримувати нові повідомлення. У разі успішного результату, на основі отриманих параметрів розраховується поточна вага вузла, чим вона більша, тим більша ймовірність того, що новий запит потрапить саме на цей вузол. Окрім того, в подібних алгоритмах розрахунку ваги існує певний ліміт, іншими словами, якщо нова розрахована вага не суттєво відрізняється від попереднього значення, то вона не буде встановлена як поточна. Це необхідно для того, аби уникнути занадто частих змін у налаштуваннях, що може призвести до перевантажень усієї системи. Також завдяки подібним алгоритмам можливо відповісти, чи потрібно додавати в систему новий вузол. Якщо вага усіх вузлів на сервері наближена до певного фіксованого значення w, то тоді це вказує на перенавантаження і необхідність масштабування системи. Це працює і в зворотній бік, якщо значення ваги на всіх вузлах високе, то це говорить про стабільну роботу системи. Для аналізу даного алгоритму він був порівняний з алгоритмом найменших з'єднань і в результаті показав кращі значення часу відповіді серверу.

Усі ці дослідження показали свою ефективність у порівнянні з базовими алгоритмами RR або мінімальних з'єднань, однак, всі вони мають певний недолік, оскільки розраховують лише вагу вузлів, але не беруть до уваги зміст вхідних запитів. Самі по собі вхідні запити відрізняються за складністю і тому на їх виконання необхідні різні ресурси і різний час. Доцільно також розподіляти складні запити на відповідні вузли, оскільки не виключено, що навіть легко навантажений вузол може швидко перенавантажитись через складність запитів на ньому. У зв'язку з цим варто також розглянути алгоритм, який вираховує можливу тривалість вхідного запиту і на цій основі передає його на відповідний вузол [7]. Найперше, що виконується в даному алгоритмі, це розрахунок очікуваного часу обробки вхідного запиту. Досягається це шляхом використання онлайн системи машинного навчання. Метод онлайнового машинного навчання – це підтип машинного навчання, у якому модель в режимі реального часу оброблює дані та надає результат. Перевагою його використання в даному випадку є те, що непотрібно зберігати результат обробки, створюючи при цьому додаткове навантаження. Також сам по собі процес машинного навчання є ресурсоємним, аби викликати його кожного разу, і тому в даному алгоритмі процес розрахунку очікуваної тривалості запитів розділений з самим процесом навчання. Це необхідно для того, аби система не витрачала ресурси на це. В алгоритмі обирається певний проміжок часу і для запитів у ньому машинне навчання розраховує потенційну тривалість, це відбувається паралельно до того, як на основі минулого проміжку і вже отриманих результатів відбувається розподілення запитів до вузлів.

Для обчислення ймовірної тривалості запиту модель машинного навчання аналізує представлення цього запиту в базі даних. По своїй суті, кожен запит в систему представляє собою операцію CRUD з даними, які містяться всередині БД [8]. Це може бути як запит на створення, читання, редагування або видалення інформації, в БД ці операції відображаються мовою SQL. Модель машинного навчання аналізує запит SQL та вибирає з нього чотири параметри:

– до якої таблиці направлений запит;
– які умови вибірки з цієї таблиці;
– чи присутні в запиті регулярні вирази;
– чи всередині запиту міститься запит до іншої таблиці.

На основі перших трьох пунктів можливо отримати очікуваний час виконання запиту і додаток до цього, завдяки останньому пункту можна також зрозуміти його складність. Обчисливши очікуваний час, даний алгоритм також аналізує стандартні для багатьох інших алгоритмів такі параметри як: навантаження на вузлі та кількість поточних з'єднань, і на основі всієї отриманої інформації розподіляє запити на відповідні вузли. Аналіз даного підходу показав ефективність у зменшенні часу обробки у порівнянні з базовими алгоритмами RR та мінімальних з'єднань.

Постановка завдання. Метою даного дослідження є аналіз недоліків наявних алгоритмів та пропозиція удосконаленого алгоритму балансування вузлів з використанням машинного навчання та динамічного обчислення ваги кожного вузла, що визначає його поточну обчислювальну спроможність, завантаженість та здатність до обробки додаткових задач у динамічних умовах. Це дозволить підвищити ефективність розподілу ресурсів, зменшити час обробки запитів та покращити загальну продуктивність системи за рахунок адаптивного коригування навантаження на кожен вузол у режимі реального часу.

Виклад основного матеріалу. Аналіз алгоритмів балансування навантаження вузлів в розподілених системах обробки великих даних базується на оцінці ключових метрик продуктивності, таких як середній час відгуку, рівень завантаженості вузлів, пропускна здатність та масштабованість.

Алгоритм Round Robin демонструє швидку обробку запитів у рівномірних середовищах з подібними вузлами, забезпечуючи середній час відгуку близько 20–30 мс у системах із рівномірним навантаженням, проте в умовах неоднорідного середовища цей показник може зростати на 50–70 %. Least Connections дає кращі результати в умовах змінного навантаження, зменшуючи середній час відгуку на 15–25 % у порівнянні з Round Robin, але збільшує затримки через додатковий моніторинг стану вузлів. Weighted Round Robin у великих системах забезпечує зниження дисбалансу до 10 %, але ефективність залежить від правильного налаштування ваг. Least Response Time може зменшити час відгуку на 20–40 % у порівнянні з простими алгоритмами, однак потребує постійного вимірювання продуктивності вузлів, що призводить до додаткових обчислювальних витрат. Consistent Hashing добре працює у масштабованих середовищах, дозволяючи знизити кількість перенаправлених запитів на 70–80 % під час змін у кластері, проте ефективність розподілу навантаження залежить від рівномірності хеш-функції. Централізоване балансування навантаження показує високу продуктивність у малих кластерах (до 100 вузлів), але зростання затримок до 30–50 % при розширенні системи робить його менш ефективним для великих кластерів. Децентралізовані методи забезпечують кращу масштабованість, але мають вищі витрати на координацію вузлів. Використання алгоритмів на основі штучного інтелекту дозволяє досягти приросту продуктивності на 15–35 % у динамічних середовищах за рахунок глибокого аналізу поточного навантаження та прогнозування майбутніх змін, хоча такі методи вимагають значних ресурсів для навчання та реалізації. Таким чином, вибір оптимального алгоритму залежить від вимог до продуктивності, масштабованості та характеристик навантаження системи, а поєднання декількох методів дозволяє досягти найкращих показників ефективності (Таблиця 1).

Алгоритми балансування навантаження вузлів використовуються у відомих програмних системах розподіленої обробки великих даних для підвищення ефективності обчислень та рівномірного розподілу ресурсів. У системі Apache Hadoop YARN застосовується комбінація Least Loaded та Fair Scheduling, що дозволяє рівномірно розподіляти обчислювальні завдання між вузлами та забезпечує мінімальний час очікування для критичних задач. Apache Spark використовує DAG Scheduler у поєднанні з Dynamic Resource Allocation, що оптимізує балансування навантаження між кластерами, дозволяючи підвищити пропускну здатність на 30–50 % у порівнянні зі статичним розподілом ресурсів. У Kubernetes, який забезпечує оркестрацію контейнеризованих додатків, застосовуються алгоритми Least Request і Weighted Round Robin, що дозволяє ефективно розподіляти навантаження між сервісами та динамічно масштабувати поділ ресурсів. Google Cloud Load Balancer використовує методи Consistent Hashing, що мінімізує кількість перенаправлених запитів під час змін у кластері, що є критично важливим для масштабованих мікросервісних архітектур. У Netflix застосовують алгоритми Adaptive Load Balancing, які базуються на машинному навчанні для передбачення навантаження та динамічного розподілу запитів між

Таблиця 1

Аналіз алгоритмів балансування навантаження вузлів

Алгоритм	Переваги	Недоліки
Round Robin	Простота реалізації, рівномірний розподіл навантаження, швидкість прийняття рішення	Не враховує продуктивність вузлів, неефективний при неоднорідному навантаженні
Least Connections	Враховує поточне навантаження вузлів, адаптується до змін у навантаженні	Вимагає додаткових обчислень для аналізу стану вузлів
Weighted Round Robin	Враховує продуктивність вузлів, ефективніший за класичний Round Robin	Потребує правильного налаштування ваг для вузлів, складніший у реалізації
Least Response Time	Мінімізує затримки обробки запитів, динамічно адаптується до змін	Вимагає постійного моніторингу часу відгуку вузлів
Consistent Hashing	Зменшує кількість переміщень даних при зміні кількості вузлів, добре підходить для кешування	Вимагає додаткових обчислень для розрахунку хешу, не завжди забезпечує рівномірний розподіл
Centralized Load Balancing	Глобальне бачення системи, можливість оптимального розподілу навантаження	Один вузол є «вузьким місцем», може стати точкою відмови
Distributed Load Balancing	Висока масштабованість, відсутність центральної точки відмови	Складність реалізації та координації між вузлами
AI/ML-based Load Balancing	Автоматична адаптація до змін у системі, оптимізація розподілу навантаження	Велика обчислювальна складність, потребує навчання моделі

серверами, що дозволяє зменшити час очікування до 40 % у пікові години навантаження. У системах обробки потокових даних, таких як Apache Kafka, використовується Partition-Aware Load Balancing, що забезпечує рівномірний розподіл повідомлень між брокерами та знижує ризик перевантаження окремих вузлів. Таким чином, різні алгоритми балансування навантаження знаходять застосування у провідних програмних платформах для забезпечення ефективності, надійності та масштабованості обробки великих даних.

Не дивлячись на наведені результати порівняння та покращення таких показників, як час обробки запиту та швидкість відповіді серверу, питання ефективності даних алгоритмів все ще відкрите. Всі вони фокусуються лише на одному або декількох параметрах, не охоплюючи загальну картину. Дуже часто при зменшенні часу відповіді значно зростає утилізація ресурсів чи вартість підтримки самої системи, що знецінює ефективність алгоритму. Іншим недоліком є те, що ці алгоритми порівнювались лише з базовими алгоритмами (RR або мінімальних з'єднань), що не є об'єктивно, оскільки дані алгоритми вже не є актуальними у сучасних системах обробки великих даних. Також в розглянутих наукових роботах зазначається, що випробування проводились у експериментальних умовах без тестування в реальних програмних системах обробки великих даних. Також сучасні алгоритми балансування вузлів майже не піднімають питання розподіленої обробки великих даних, про що зазначено в дослідженнях [9]. Наведені приклади не є виключенням, навіть при умовах лабораторних досліджень не проводилась симуляція генерування великих даних, і тому цей аспект лишається відкритим.

У зв'язку з розглянутими недоліками, у даній роботі планується розробити удосконалений алгоритм балансування вузлів з використанням машинного навчання та динамічного обчислення ваги з подальшим тестуванням отриманого результату на великих даних. Причина використання машинного навчання полягає у тому, що на основі проведеного дослідження даний підхід показав найкращі результати у порівняні з алгоритмами RR та мінімальних з'єднань. Недоліком цього підходу є те, що машинне навчання потребує додаткових ресурсів для своєї роботи. В самому дослідженні був акцент на цьому аспекті, однак, не зазначено чи покращення результатів вартувало росту використання ресурсів. Через використання динамічного обчислення ваги планується обійти даний недолік, оскільки подібні алгоритми не вимагають надлишкових ресурсів і, не дивлячись на нижчу ефективність у порівнянні з використанням машинного навчання, все одно показали покращення ключових параметрів. Через поєднання цих двох підходів планується досягти балансу між використанням можливостей машинного навчання та покращити ефективність динамічного розрахунку ваги.

У запропонованому алгоритмі після надходження нового запиту система буде розраховувати вагу кожного з вузлів, після чого буде розраховуватися очікувана тривалість цього запиту з урахуванням отриманої ваги і вже на основі отри-

маного результату відбуватиметься розподіл між доступними варіантами (для кращого розуміння вважаєм, що сервер має два вузли). Даний метод потребує експериментального дослідження, адже теоретично неможливо прорахувати його ефективність, тільки дослідження з живим набором даних може дати точні результати.

Запропонований алгоритм вибору вузла для обробки запиту базується на розрахунку метрик продуктивності кожного доступного вузла. Після надходження запиту система аналізує такі характеристики вузлів, як поточне завантаження, швидкість обробки та інші параметри, і на їх основі обчислює вагові коефіцієнти для кожного вузла. Далі виконується розрахунок очікуваної тривалості виконання запиту на кожному вузлі. На основі цих значень алгоритм приймає рішення про розподіл запиту, вибираючи найбільш оптимальний вузол (N1 або N2) для обробки, що мінімізує час виконання та забезпечує рівномірне балансування навантаження.

Висновки. Аналіз алгоритмів балансування вузлів у розподілених програмних системах обробки великих даних показав, що кожен із розглянутих підходів має свої переваги та обмеження залежно від конкретного сценарію використання. Least Loaded та Fair Scheduling забезпечують рівномірний розподіл навантаження, але можуть не враховувати специфіку оброблюваних задач. DAG Scheduler оптимізує виконання складних багатозалежних процесів, однак його складність обмежує масштабованість. Алгоритм Dynamic Resource Allocation ефективний для адаптивних систем, проте потребує додаткових обчислювальних ресурсів для моніторингу.

Алгоритми Least Request та Weighted Round Robin добре працюють у високонавантажених сервісах, але можуть страждати від нестабільності при різкій зміні навантаження. Consistent Hashing зменшує витрати на перенаправлення запитів, проте менш ефективний для рівномірного балансування. Adaptive Load Balancing та Partition-Aware Load Balancing забезпечують високу продуктивність у масштабованих системах, але вимагають складної конфігурації.

Результати дослідження показали, що комбіновані або гібридні підходи, що поєднують адаптивність та контекстну оптимізацію, мають найбільший потенціал для використання в сучасних розподілених середовищах. Усі проаналізовані методи мали обмежене тестування з експериментальним набором даних, порівнювались лише з базовими застарілими алгоритмами балансування, такими як RR та мінімальних з'єднань і фокусувались лише на небагатьох параметрах, ігноруючи значення інших. Крім того, в даних алгоритмах не було розкрите питання розподіленої обробки великих даних та ефективності роботи з ними. У зв'язку з цим виникає потреба у розробці алгоритму балансування вузлів з використанням найкращих практик існуючих алгоритмів, таких як машинне навчання та розрахунок ваги. Для вдосконаленого алгоритму планується виконати тестування з урахуванням недоліків попередніх підходів. Планується порівняння даного алгоритму з сучасними аналогами та дослідженням його ефективності при роботі з великими даними з урахуванням значень усіх параметрів ефективності. Незважаючи на розвиток алгоритмів балансування, існують проблеми, пов'язані з високою варіативністю робочих навантажень, необхідністю швидкої адаптації та ефективного управління ресурсами у розподілених системах обробки великих даних. Перспективними напрямами досліджень є використання глибокого навчання для прогнозування навантаження, інтеграція балансування з технологіями Edge Computing для розподілених обчислень на периферії мережі, а також оптимізація енергоспоживання в дата-центрах через адаптивні алгоритми балансування.

Список літератури:

1. Zhou S. An Improved dynamic load-balancing Algorithm for Cluster. *Journal of Computer and Modernization*. 2012. Vol. 1. P. 135–139. DOI: 10.3969/j.issn.1006-2475.2012.01.036

2. Singh N., Hamid Y., Juneja S. Load balancing and service discovery using Docker Swarm for microservice based big data applications. *Journal of Cloud Computing 12, 4*. 2023. DOI: https://doi.org/10.1186/s13677-022-00358-7

3. Pan Z., Jiangxing Z. Load Balancing Algorithm for Web Server Based on Weighted Minimal Connections. *Journal of Web Systems and Applications*. 2017. Vol. 1. P. 1–8. DOI: https://dx.doi.org/10.23977/jwsa.2017.11001

4. Pei-rui J., Li-min M., Yu-zhou S., Yang-tian-xiu H. A client proximity based load balance algorithm in web sever cluster. *2nd International Conference on Wireless Communication and Network Engineering*. 2017. P. 317–322.

5. Li R., Li Y., Li W. An integrated load-balancing scheduling algorithm for nginx-based web application clusters. *Journal of Physics: Conference Series 1060 012078*. 2018. DOI: 10.1088/1742-6596/1060/1/012078

6. Wei C., Hou J., Ma D., Zhao J., Sun Y. Design and implementation of a TCP long connection load balancing algorithm based on negative feedback mechanism. *Journal of Physics: Conference Series 1659 012001*. 2020. DOI: 10.1088/1742-6596/1659/1/012001

7. Omori M., Nishi H. Request Distribution for Heterogeneous Database Server Clusters with Processing Time Estimation. *International Conference on Industrial Informatics* (INDIN), Porto. 2018, P. 278–283. DOI: 10.1109/INDIN.2018.8471931

8. Truica C.-O., Radulescu F., Boicea A., Bucur I.. Performance Evaluation for CRUD Operations in Asynchronously Replicated Document Oriented Database. *International Conference on Control Systems and Computer Science*. Bucharest. 2015. P. 191–196, DOI: 10.1109/CSCS.2015.32

9. Jafarnejad Ghomi E., Masoud Rahmani A., Nasih Qader N.. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications*. Vol. 88. 2017. P. 50–71. DOI: https://doi.org/10.1016/j.jnca.2017.04.007.

Vovchenko D. S., Oleshchenko L. M. ANALYSIS OF NODE BALANCING ALGORITHMS IN DISTRIBUTED BIG DATA PROCESSING SOFTWARE SYSTEMS

Load balancing algorithms play a crucial role in distributed big data processing systems by ensuring efficient resource utilization, reducing query execution time, and enhancing system reliability. The choice of a specific algorithm depends on system characteristics, workload type, and performance requirements. This paper presents current algorithmic solutions to the load balancing problem in multi-node systems. The research examines existing node balancing algorithms, including Least Loaded, Fair Scheduling, DAG Scheduler, Dynamic Resource Allocation, Least Request, Weighted Round Robin, Consistent Hashing, Adaptive Load Balancing, and Partition-Aware Load Balancing, aimed at improving the overall efficiency of distributed big data processing systems.

The objective of this paper is to investigate the effectiveness of modern solutions and highlight unresolved timeliness issues: the predominant focus on common parameters such as server response time and query processing time while ignoring other efficiency factors such as resource utilization and cost, as well as limitations in research that overlook the specifics of big data processing. The analysis of load balancing algorithms in distributed big data processing systems has shown that methods such as Dynamic Resource Allocation in Apache Spark can increase throughput by 30–50 %, Adaptive Load Balancing in Netflix reduces request wait time by up to 40 % during peak periods, and Consistent Hashing in Google Cloud Load Balancer minimizes request redirection when cluster configurations change, thereby enhancing system stability. The main drawbacks of the analyzed algorithms include limited adaptability to unpredictable workload changes, delays in scaling, as well as complexity in configuration and computational overhead for dynamic resource redistribution.

Future research aims to develop a node balancing algorithm that takes into account all the advantages and disadvantages of modern solutions. The proposed improved node balancing algorithm will prevent the issue of overloading specific nodes and demonstrate the feasibility of using dynamic node weight calculation technologies and machine learning in modern systems, considering a broader range of efficiency parameters and big data processing requirements.

Key words: *big data, distributed multi-node software systems, node balancing algorithms, CPU, server memory, database operations, machine learning.*