УДК 004.738.5:004.056.53
DOI https://doi.org/10.32782/2663-5941/2024.6.2/03

Боровскова Є.А.
AppsFlyer Ltd

ВИЗНАЧЕННЯ ОПТИМАЛЬНИХ СТРАТЕГІЙ КЕШУВАННЯ ДЛЯ ПІДВИЩЕННЯ ПРОДУКТИВНОСТІ СЕРВЕРНИХ ДОДАТКІВ НА NESTJS

Стаття присвячена дослідженню продуктивності серверних додатків у сучасних інформаційних системах, що працюють із великими обсягами даних у режимі реального часу. Розкрито необхідність підвищення продуктивності для забезпечення стабільної роботи таких додатків за умов підвищеного навантаження, що є критично важливим для оптимізації використання ресурсів та скорочення часу обробки запитів. Встановлено, що є управління кешем є однією з ефективних стратегій, яка зменшує кількість звернень до бази даних, знижує навантаження на процесор і скорочує час обробки запитів.

У статті також висвітлено сучасні підходи до кешування та визначено їхній вплив на продуктивність серверних систем, що базуються на фреймворку NestJS. У процесі дослідження використано методи тестування навантаження, зокрема Apache JMeter, для моделювання високонавантажених середовищ, а також інструменти моніторингу, такі як Datadog, для контролю ефективності кешування в реальному часі. З'ясовано, що впровадження кешування зменшує середній час обробки запитів на 90%, знижує навантаження на процесор наполовину та скорочує кількість звернень до бази даних на 95%. Це дозволяє підвищити ефективність роботи додатків і знизити витрати на інфраструктуру, зокрема завдяки зменшенню споживання оперативної пам'яті та процесорних ресурсів.

Визначено важливість правильного вибору стратегій кешування, які враховують специфіку запитів і доступні ресурси системи. Розкрито ефективність адаптивного кешування на основі аналітики реального часу та TTL-кешування, що дозволяє динамічно адаптувати систему для забезпечення максимальної ефективності. Наголошено на потребі ретельного аналізу й налаштування, щоб уникнути проблем із застарілими даними та забезпечити коректну консистентність інформації.

Стаття також містить аналіз впливу обраних стратегій кешування на оптимізацію витрат на підтримку інфраструктури. Результати підтверджують, що застосування сучасних підходів до управління кешем дозволяє досягти значного підвищення продуктивності та стабільності серверних додатків у високонавантажених умовах. Перспективи подальших досліджень полягають у вивченні інтеграції машинного навчання для автоматизації управління кешем, що сприятиме оптимізації ресурсів на основі аналізу історичних даних і передбачення навантаження на систему.

Ключові слова: *оптимізація продуктивності, серверні додатки, управління кешем, ефективність оброблення даних, стратегії збереження.*

Постановка проблеми. Проблема оптимізації продуктивності серверних додатків набуває особливого значення в умовах стрімкого зростання обсягів даних і підвищених вимог до ефективності їх оброблення. У сучасному інформаційному середовищі серверні додатки є ключовими компонентами інфраструктури, що забезпечують доступ до різноманітних сервісів і функціональних можливостей. З огляду на це оптимізація їх продуктивності є важливою задачею як з наукової, так і з практичної точки зору. Одним із важливих аспектів такої оптимізації є ефективне управління кешем, оскільки правильний підхід до кешування дозволяє значно знизити навантаження на сервери, скоротити час оброблення запитів та зменшити витрати на обчислювальні ресурси. Використання фреймворку NestJS, який є популярним для створення високопродуктивних серверних додатків, потребує визначення оптимальних стратегій кешування для підвищення його ефективності в різних умовах.

Розв'язання цієї проблеми є актуальним не лише для підвищення продуктивності окремих додатків, але й для забезпечення стабільної та ефективної роботи комплексних систем, які обслуговують великий обсяг користувачів. Це завдання має тісний зв'язок з питаннями масштабованості та надійності додатків, а також з актуальними науковими дослідженнями в галузі розподілених обчислень і управління даними.

Аналіз останніх досліджень і публікацій. Аналіз сучасних досліджень, присвячених опти-

мальним стратегіям кешування для підвищення продуктивності серверних додатків, охоплює різноманітні підходи, зокрема порівняння фреймворків, вибір стратегій кешування та автоматизацію процесів.

Дослідження М. Прасад і У. Падма (M. Prasad, U. Padma) порівнює продуктивність ExpressJS і Fastify в контексті фреймворку NestJS, підкреслюючи важливість вибору фреймворку для досягнення високої продуктивності серверних додатків [1]. Це дослідження доповнюється роботою Б. Зіма і М. Барщ (B. Zima, M. Barszcz), які порівнюють різні Node.js фреймворки, що дозволяє вибрати найоптимальніший варіант залежно від умов використання [2]. Аналогічно дослідження М. Голець і М. Плехавської-Войчик (M. Golec, M. Plechawska-Wójcik) підкреслює значення використання TypeScript фреймворків для побудови серверних додатків і підвищення їх продуктивності через оптимізацію кешування [3].

Також предметом сучасних досліджень є стратегії кешування. М. Зульфа, Р. Хартанто і А. Перманасарі (M. Zulfa, R. Hartanto, A. Permanasari) у своєму систематичному огляді аналізують різні стратегії кешування для вебдодатків, наголошуючи на важливості оптимального управління кешем для зменшення навантаження на сервери та покращення продуктивності [4]. Це узгоджується з дослідженням М. Заеда (M. Zahed) та співавторів, де розглянуто «зелені» стратегії кешування для комунікаційних мереж нового покоління з акцентуванням на зменшенні енергоспоживання без втрати продуктивності [5].

М. Наїм (M. Naeem) з колегами вивчає різні стратегії кешування популярного контенту в мережах Named Data Networking (NDN), аналізуючи порівняльну ефективність стратегій кешування, а також пропонує складні стратегії для оптимізації кешування контенту, що дозволяє суттєво підвищити продуктивність систем у реальних умовах [6; 7]. Це підкреслює важливість вибору стратегії кешування для забезпечення ефективного використання ресурсів.

Л. Чжао (L. Zhao) та інші досліджують інтелектуальні стратегії кешування для автономного транспорту, зокрема в системах 6G, де використання таких підходів дозволяє суттєво підвищити ефективність оброблення даних [8]. Дж. Шуджа (J. Shuja) та інші акцентують увагу на автоматизації процесів кешування через машинне навчання, що дозволяє системам динамічно адаптуватися до умов використання, підвищуючи продуктивність і гнучкість серверних додатків [9].

С. Чен (C. Chen et al.) та інші у своєму дослідженні акцентують увагу на інтелектуальних стратегіях кешування, що враховують часово-просторові характеристики для покращення продуктивності в транспортних мережах [10]. А. Бекман (A. Beckmann et al.) та співавтори підкреслюють важливість використання цифрових систем для підвищення продуктивності через інтеграцію кешування в середовищі спільної роботи [11]. Т. Кушевич і Д. Блажевич (T. Kušević, D. Blažević) порівнюють функціональні можливості протоколів HTTP і MQTT, що також впливає на вибір стратегії кешування та оптимізацію використання серверних ресурсів [12]. Нарешті, дослідження Ч. Лі (C. Li) та співавторів присвячено стратегіям кешування в мобільних обчислювальних системах, підкреслено важливість оптимізації затримок та енергоспоживання [13].

Ці дослідження підкреслюють важливість правильного вибору фреймворків та стратегій кешування для досягнення максимальної ефективності серверних додатків, зокрема через автоматизацію процесів і застосування інтелектуальних рішень.

Виділення не вирішених раніше частин загальної проблеми, котрим присвячується означена стаття. Попри велику кількість досліджень у сфері кешування серверних додатків, залишаються не розв'язаними питання щодо детального аналізу сучасних підходів та їх реального впливу на продуктивність. Окрім загальних принципів кешування, необхідно глибше дослідити ключові фактори, що впливають на швидкість виконання HTTP-запитів у таких фреймворках, як NestJS, зокрема в умовах навантаженого середовища. Вивчення цього аспекту може бути підкріплено емпіричними дослідженнями із застосуванням інструментів для тестування навантаження, що дозволить оцінити ефективність різних стратегій кешування в реальних умовах. На основі отриманих даних доцільно розробити оптимальні стратегії, орієнтовані на підвищення продуктивності додатків та ефективного моніторингу їх роботи, що стане вагомим внеском у практику адміністрування та підтримки серверних систем.

Постановка завдання. Мета статті полягає в дослідженні та обґрунтуванні оптимальних стратегій кешування для підвищення продуктивності серверних додатків на основі фреймворку NestJS з акцентом на покращення швидкості оброблення запитів і ефективності використання ресурсів.

Завдання статті:

1. Проаналізувати сучасні підходи до кешування в серверних додатках та їх вплив на продуктивність;

2. Визначити ключові фактори, що впливають на швидкість виконання HTTP-запитів у середовищі NestJS з використанням кешування;

3. Провести емпіричне дослідження з використанням інструментів тестування навантаження для оцінювання ефективності вибраних стратегій;

4. Надати рекомендації щодо впровадження оптимальних стратегій кешування для підвищення продуктивності та ефективного моніторингу серверних додатків.

Виклад основного матеріалу. Оптимізація продуктивності серверних додатків є важливим завданням у сучасних інформаційних системах, оскільки вона безпосередньо впливає на швидкість оброблення запитів і ефективність використання ресурсів. Одним із найпоширеніших способів підвищення продуктивності є кешування, яке дозволяє зберігати результати виконаних запитів для їх повторного використання.

Оцінювання потенційних економічних вигод від застосування кешування в серверних додатках є важливим аспектом аналізу його ефективності. Зниження навантаження на інфраструктуру та оптимізація використання серверних ресурсів можуть призвести до значного скорочення витрат, що особливо актуально в умовах великих навантажень на систему. Використання кешування дозволяє значно зменшити кількість звернень до бази даних, що у свою чергу знижує необхідність у дорогих обчислювальних потужностях для підтримки високої швидкості оброблення запитів. Менша кількість операцій введення – виведення та зниження завантаженості бази даних можуть скоротити витрати на масштабування серверів або додаткові ресурси для підтримки стабільної роботи під час пікових навантажень [1].

Економічні вигоди також проявляються через підвищення масштабованості системи. Завдяки зниженню споживання процесорних ресурсів та оперативної пам'яті під час кешування серверні додатки можуть обробляти більшу кількість запитів без необхідності збільшення кількості фізичних або віртуальних серверів. Це дозволяє знизити витрати на хостинг, оренду або підтримку додаткових серверних потужностей, що є значною статтею витрат у середовищі, де продуктивність та швидкість відповіді мають вирішальне значення для користувачів.

Крім того, скорочення часу оброблення запитів дозволяє забезпечити кращий користувацький досвід, що є важливим економічним чинником для компаній, які надають онлайн-послуги або працюють у сфері електронної комерції. Швидше оброблення запитів та зменшення затримок можуть збільшити лояльність користувачів і покращити конверсію, що безпосередньо впливає на дохід [2; 3]. Таким чином, економічний ефект від впровадження кешування не обмежується лише оптимізацією витрат на інфраструктуру, але й сприяє підвищенню рентабельності бізнесу загалом через покращення показників продуктивності.

Крім того, зниження енергоспоживання серверів унаслідок зменшення навантаження на процесорні ресурси також може стати важливим економічним фактором. Сервери, що працюють з меншими навантаженнями, відповідно, споживають менше енергії, що знижує витрати на підтримку роботи дата-центрів і одночасно сприяє екологічній стійкості компаній, які впроваджують такі технології. У довгостроковій перспективі це може стати важливим чинником зменшення операційних витрат, особливо для великих організацій з великим обсягом оброблення даних.

Таким чином, кешування не лише оптимізує технічні параметри серверної інфраструктури, але й надає реальні економічні вигоди завдяки скороченню витрат на інфраструктуру, підвищенню масштабованості системи та покращенню користувацького досвіду.

Сучасні підходи до кешування поділяються на кілька рівнів, включаючи клієнтське кешування, серверне кешування, а також проксі-кешування. Важливою частиною ефективного кешування є вибір правильної стратегії зберігання та оновлення даних. Це включає такі підходи, як: кешування на основі часу життя даних (TTL); адаптивне кешування, де час життя даних залежить від частоти їх запитів; стратегія кешування «лінивого завантаження» (lazy loading), де дані кешуються тільки після першого запиту.

У таблиці 1 продемонстровано ключові характеристики різних підходів до кешування, їх переваги та недоліки в контексті використання серверних додатків, що базуються на фреймворці NestJS.

Використання відповідної стратегії в серверних додатках, що базуються на фреймворці NestJS, дозволяє значно підвищити продуктивність системи. Наприклад, серверне кешування знижує кількість звернень до бази даних, що є критично важливим для високонавантажених додатків, де кожен запит до бази може займати значний час. На практиці це дозволяє зменшити час відповіді на запити з кількох секунд до мілісекунд, що робить систему більш чутливою та ефективною. Інструменти тестування, як-от Apache JMeter, можуть бути використані для вимірювання цих

Таблиця 1

Ключові характеристики різних підходів до кешування, що базуються на фреймворці NestJS

Підхід до кешування	Переваги	Недоліки	Застосування на практиці
Клієнтське кешування	Зменшує кількість звернень до сервера; швидкий доступ до даних	Дані можуть бути застарілими	Часто використовується для зберігання статичних ресурсів, як-от зображення та стилі
Серверне кешування	Зменшує навантаження на сервер і базу даних, прискорює оброблення запитів	Вимагає додаткових ресурсів для керування кешем	Використовується для збереження результатів складних запитів до бази даних
Проксі-кешування	Дозволяє зберігати дані на проміжних серверах, що знижує навантаження на основний сервер	Може призводити до зберігання застарілих даних	Ефективно для масштабованих систем з великим обсягом запитів
Кешування на основі TTL	Встановлює конкретний час життя даних у кеші, що забезпечує їх актуальність	Може спричиняти видалення даних до повторного запиту	Використовується для даних, які потребують регулярного оновлення
Адаптивне кешування	Оптимізує використання кешу на основі частоти доступу до даних	Складна реалізація, потребує аналізу частоти запитів	Корисно для зменшення обчислювальних витрат для рідко використовуваних даних
Lazy loading (ліниве завантаження)	Дані кешуються тільки після першого запиту, що зменшує початкове навантаження на сервер	Перший запит може бути повільним, оскільки відсутнє попереднє кешування	Використовується для систем, де початкове навантаження на сервер має бути мінімальним

Джерело: сформовано автором на підставі [5; 6; 7; 8]

показників продуктивності в реальних умовах, тоді як системи моніторингу, зокрема Datadog, забезпечують постійний аналіз стану системи та дозволяють вчасно виявляти можливі проблеми або «вузькі місця» в роботі серверних додатків.

Кешування на основі TTL надає можливість налаштувати автоматичне видалення даних з кешу через певний проміжок часу, що дозволяє підтримувати баланс між актуальністю даних і ефективністю системи. Практично це особливо важливо для додатків, які працюють з динамічними даними, що часто оновлюються.

Швидкість виконання HTTP-запитів у серверних додатках на базі NestJS значною мірою залежить від низки факторів, які впливають на ефективність використання кешу. Основними аспектами, що визначають продуктивність системи, є конфігурація кешування, розмір і складність запитів, а також апаратні ресурси, доступні для виконання операцій. NestJS дозволяє впроваджувати різні стратегії кешування, включаючи глобальне кешування для всього додатку або вибіркове кешування на рівні окремих маршрутів чи запитів, що безпосередньо впливає на швидкість оброблення запитів.

Ефективність кешування також залежить від характеру запитів, які можуть бути ідентичними або подібними. У разі часто повторюваних запитів кешування дозволяє уникнути повторного оброблення тих самих даних, що значно прискорює відповіді на запити. Однак за складних запитів або запитів до великих обсягів даних кешування може бути менш ефективним, якщо обсяги даних, що зберігаються в кеші, перевищують доступні ресурси. Інший важливий аспект – це вибір стратегії кешування. Наприклад, використання кешування на основі часу життя даних (TTL) дозволяє контролювати актуальність збереженої інформації, проте це може призводити до того, що в разі повторних запитів застарілі дані будуть видалені з кешу, і запит доведеться обробляти заново. З іншого боку, надмірно тривале зберігання даних у кеші може перевантажувати сервер обмеженими ресурсами пам'яті.

У таблиці 2 проілюстровано основні фактори, що впливають на швидкість виконання HTTP-запитів із використанням кешування в середовищі NestJS.

Вплив кешування на продуктивність серверних додатків можна більш точно оцінити через детальне дослідження ключових факторів, що визначають швидкість виконання HTTP-запитів [10]. З огляду на специфіку фреймворку NestJS кожен із зазначених у таблиці факторів по-різному

Таблиця 2

Ключові фактори, що впливають на швидкість виконання HTTP-запитів із використанням кешування в середовищі NestJS

Фактор	Вплив на продуктивність	Опис
Конфігурація кешування	Високий	Вибір глобального чи вибіркового кешування впливає на обсяг даних, які можна зберігати в кеші, що безпосередньо впливає на швидкість оброблення запитів
Характер запитів	Середній	Частота та ідентичність запитів визначають, наскільки ефективно система може використовувати дані, що зберігаються в кеші для повторного оброблення
Складність запитів	Високий	Більш складні запити, що вимагають доступу до великих обсягів даних, можуть уповільнювати продуктивність системи через обмеження кешу
Стратегія збереження в кеші	Середній	Використання TTL або адаптивного кешування може вплинути на ефективність системи залежно від частоти звернень та актуальності даних
Апаратні ресурси	Високий	Кількість доступної пам'яті та потужність процесора визначають, скільки даних можна зберігати в кеші та наскільки швидко обробляються запити

Джерело: сформовано автором

впливає на загальну продуктивність. Конфігурація кешування, наприклад, визначає, як дані будуть зберігатися і використовуватися: глобальна стратегія кешування охоплює всі маршрути, тоді як вибіркове кешування дозволяє зберігати дані лише для певних маршрутів, що дає змогу краще оптимізувати ресурси для часто використовуваних даних.

Аналіз характеру запитів також розкриває важливий аспект продуктивності. Частота звернень до одних і тих самих ресурсів напряму впливає на доцільність кешування: для часто повторюваних запитів виграш у продуктивності буде значним, тоді як для унікальних або складних запитів кешування може бути менш ефективним або навіть зайвим.

Особлива увага приділяється стратегіям збереження в кеші, адже використання таких механізмів, як TTL (час життя кешованих даних), дозволяє підтримувати актуальність інформації, але вимагає ретельного налаштування. Недостатньо гнучка стратегія може призводити до затримок або до видалення необхідних даних з кешу передчасно, що вплине на час оброблення нових запитів.

У реальних умовах вплив цих факторів на продуктивність залежить також від апаратних ресурсів. Кількість доступної пам'яті та процесорної потужності безпосередньо визначає ефективність роботи кешу. Недостатня кількість оперативної пам'яті може призвести до ситуації, коли кешування не забезпечить очікуваного підвищення продуктивності або навіть погіршить ситуацію через часте очищення кешу [12; 13].

Сучасні дослідження в галузі кешування підкреслюють важливість ефективної інтеграції цієї технології для підвищення продуктивності серверних додатків. Наприклад, у фреймворці NestJS реалізація кешування дозволяє значно зменшити навантаження на базу даних та прискорити оброблення запитів [14]. Використання кешування у Node.js, зокрема через такі технології, як Redis, демонструє значні переваги в зниженні навантаження на систему [15, 16]. Одним із важливих інструментів, який дозволяє легко інтегрувати кешування в серверні додатки, є npm-пакет cache-manager, що забезпечує високу гнучкість у виборі стратегії кешування [17]. Крім того, для моніторингу та аналізу продуктивності систем у реальному часі використовуються такі платформи, як Datadog, які дозволяють контролювати стан серверів і виявляти потенційні вузькі місця [18]. API, розроблені на базі NestJS, також можуть бути оптимізовані завдяки використанню адаптивних підходів до кешування, що значно покращує швидкість виконання HTTP-запитів [19].

Інструменти для тестування навантаження, як-от Apache JMeter, забезпечують моделювання умов високого навантаження, дозволяючи в реальних умовах оцінювати ефективність упровадженого кешування [20; 21]. Як демонструють дослідження, використання Redis для кешування запитів допомагає оптимізувати не лише продук-

тивність серверних додатків, а й знижує витрати на підтримку інфраструктури [22]. Такі підходи підкреслюють важливість налаштування ефективних алгоритмів кешування, які можуть значно мінімізувати затримки і покращити загальну продуктивність систем [23].

З метою оцінювання впливу кешування на продуктивність серверних додатків, побудованих на фреймворці NestJS, було проведено емпіричне дослідження. Для цього було використано інструменти тестування навантаження та моніторингу, які забезпечили збір точних даних про швидкість оброблення HTTP-запитів і використання системних ресурсів. Тестування проводилося у двох режимах: без застосування кешування та з його використанням. Такий підхід дозволив виявити відмінності в продуктивності та ефективності оброблення запитів, що було проаналізовано на основі зібраних показників.

Експеримент відбувався в умовах високого навантаження на сервер, що дозволило отримати релевантні дані для оцінювання реальних сценаріїв використання. Запитів до сервера було здійснено достатньо для точного порівняння часу оброблення та рівня споживання ресурсів. За допомогою Apache JMeter було забезпечено одночасну генерацію великої кількості запитів, що імітувало високий трафік, а Datadog дозволив відслідковувати зміни в продуктивності додатка в режимі реального часу (табл. 3).

Дані, отримані під час експерименту, чітко демонструють ефективність кешування в підвищенні продуктивності додатка. Без кешування середній час виконання HTTP-запитів становив близько 1,5 секунд, тоді як із застосуванням кешу цей час скоротився до 0,15 секунди, що становить приблизно 90% зменшення. Це свідчить про значне покращення швидкості оброблення запитів, що особливо важливо в умовах високого навантаження на сервер.

Крім того, кількість звернень до бази даних зменшилася майже на 95%, що знижує навантаження на серверні ресурси і дозволяє оптимізувати роботу системи. Рівень використання центрального процесора також знизився майже наполовину, що дозволяє зробити висновок про більш ефективне використання ресурсів під час кешування. Споживання оперативної пам'яті зменшилося на третину, що свідчить про стабілізацію використання системних ресурсів.

Ці результати показують значну ефективність кешування в контексті підвищення продуктивності серверних додатків, що робить його невід'ємною частиною стратегії оптимізації вебсервісів, особливо тих, які працюють у реальному часі. Кешування є потужним інструментом для підвищення продуктивності серверних додатків, але його ефективне впровадження вимагає комплексного підходу, що враховує специфіку застосовуваних технологій та архітектуру системи. Основними проблемами під час впровадження кешування є можливість некоректного налаштування, що може призвести до розбалансування навантаження на ресурси, неефективного використання пам'яті або до затримок через часті оновлення кешованих даних. Більш складними є питання забезпечення консистентності даних у розподілених системах, де кешування може спричинити невідповідності через асинхронність оновлень. Крім того, важливим аспектом є контроль і управління застарілими даними, що може впливати на цілісність і достовірність інформації, яку обробляє додаток [24; 25]. Ці проблеми потребують стратегічних рішень, орієнтованих на комплексну оптимізацію системи (табл. 4).

Запропоновані рекомендації відображають сучасні підходи до розроблення і впровадження стратегій кешування в умовах високонавантажених серверних додатків. Інтеграція розподіленого кешу на рівні мікросервісної архітектури дозволяє ефективно балансувати навантаження між серверами, забезпечуючи вищу масштабованість системи та її стійкість до пікових навантажень.

Таблиця 3

Порівняльний аналіз продуктивності серверного додатка на базі NestJS з використанням та без використання кешування

Параметр	Без кешування	З кешуванням	Різниця
Середній час виконання запитів	1,5 сек.	0,15 сек.	~90% зменшення
Навантаження на CPU	85%	45%	~47% зниження
Кількість звернень до бази даних	1000 запитів	50 запитів	~95% зменшення
Споживання оперативної пам'яті	1,8 ГБ	1,2 ГБ	~33% зниження

Джерело: авторська розробка

Таблиця 4

Рекомендації щодо впровадження стратегій кешування для високонавантажених серверних додатків

Рекомендація	Обґрунтування	Потенційні проблеми та стратегії їх розв'язання
Інтеграція розподіленого кешу на рівні мікросервісної архітектури	Підвищує масштабованість і знижує навантаження на окремі вузли системи	Може ускладнити забезпечення консистентності даних, що вирішується застосуванням стратегій «write-through» та «read-through»
Використання адаптивного кешування на основі аналітики реального часу	Забезпечує гнучке налаштування кешу залежно від навантаження і потреб	Вимагає інтеграції складних алгоритмів, що може збільшити навантаження на систему в моменти пікових звернень
Впровадження стратегій кешування на стороні клієнта та на рівні CDN	Знижує навантаження на сервер та оптимізує доставку контенту для глобальних користувачів	Можливі проблеми з актуальністю кешованих даних у різних географічних точках, що вирішується налаштуванням політик TTL (time-to-live)
Автоматизація управління кешем з використанням машинного навчання	Оптимізує управління ресурсами та підвищує продуктивність через автоматизовані рішення	Складність запровадження та потреба в значних ресурсах для навчання моделей, що може бути оптимізовано через поетапне впровадження і тестування

Джерело: авторська розробка

Однак така архітектура може стикатися з проблемами консистентності даних, які розв'язуються шляхом використання стратегій «write-through» та «read-through», що забезпечують узгодженість між кешем та основною базою даних.

Адаптивне кешування на основі аналітики реального часу надає можливість динамічно підлаштовувати стратегії кешування залежно від поточного стану системи і навантаження. Це дозволяє уникати надмірних оновлень кешу або недоцільного використання системних ресурсів, проте вимагає інтеграції складних алгоритмів аналітики, що може стати додатковим викликом для розробників.

Впровадження кешування на стороні клієнта і на рівні CDN значно підвищує ефективність серверних додатків, особливо в умовах глобальних мереж, де доступ до контенту може бути сповільнений через віддаленість користувача. Однак проблема актуальності кешованих даних залишається важливою, і її можна розв'язати налаштуванням точних політик часу життя кешу (TTL), що дозволяє системі автоматично очищати старі дані з кешу.

Найбільш інноваційною стратегією є використання машинного навчання для автоматизації управління кешем [9], що дозволяє системі навчатися на основі історичних даних та оптимізувати кешування в реальному часі. Це забезпечує максимальну ефективність використання ресурсів та мінімізує втрати продуктивності. Однак упровадження таких рішень є ресурсомістким процесом, що потребує поетапного підходу для успішної інтеграції та тестування ефективності алгоритмів у реальних умовах експлуатації.

Очікується, що ці стратегії дозволять значно підвищити продуктивність серверних додатків завдяки оптимізації використання ресурсів, покращення масштабованості та мінімізації часу оброблення запитів.

Висновки. Емпіричні дослідження довели, що використання стратегій кешування суттєво підвищує продуктивність серверних додатків, зокрема на фреймворці NestJS. Встановлено, що кешування зменшує середній час оброблення HTTP-запитів з 1,5 секунд до 0,15 секунди, що свідчить про приблизно 90% скорочення. Це також призводить до значного зменшення кількості звернень до бази даних і зниження навантаження на процесор, яке в середньому зменшується на 47%. Основними проблемами, що були виявлені під час дослідження, є складність у забезпеченні консистентності даних, особливо в умовах використання розподіленого кешу, а також необхідність коректного налаштування механізмів оновлення кешу для уникнення зберігання застарілої інформації. Використання адаптивного кешування і стратегій кешування на основі часу життя даних дозволяє розв'язати ці проблеми, забезпечуючи актуальність даних і оптимізацію використання ресурсів. Проте їх впровадження вимагає додаткових обчислювальних потужностей і ретельного налаштування.

Рекомендується інтеграція розподіленого кешу на рівні мікросервісної архітектури, що

дозволить підвищити масштабованість додатків і ефективно балансувати навантаження на систему. Використання стратегії «write-through» та «read-through» може мінімізувати ризики неконсистентності даних у системах з великою кількістю вузлів. Іншою перспективною стратегією є впровадження автоматизації управління кешем за допомогою машинного навчання, що дозволить оптимізувати використання кешу на основі реальних даних і прогнозів навантаження. Це також дасть змогу значно знизити витрати на підтримку інфраструктури та підвищити ефективність використання серверних ресурсів.

Подальші дослідження мають бути зосереджені на інтеграції цих стратегій в умовах динамічних середовищ, а також на дослідженні їх впливу на продуктивність серверних додатків у реальних сценаріях використання.

Список літератури:

1. Prasad M., Padma U. Performance Analysis of ExpressJS and Fastify in NestJS. In: International Conference on Robotics, Control, Automation and Artificial Intelligence. Singapore: Springer Nature Singapore. 2022. P. 1037-1049. URL: https://link.springer.com/chapter/10.1007/978-981-99-4634-1_82#citeas (date of access: 17.10.2024)

2. Zima B., Barszcz M. Comparative analysis of Node.js frameworks. *Journal of Computer Sciences Institute.* 2024. Vol. 30. P. 26-30. DOI: https://doi.org/10.35784/jcsi.5364 (date of access: 17.10.2024)

3. Golec M., Plechawska-Wójcik M. Comparative analysis of frameworks using TypeScript to build server applications. *Journal of Computer Sciences Institute.* 2022. Vol. 23. P. 128-134. DOI: https://doi.org/10.35784/jcsi.2910 (date of access: 17.10.2024)

4. Zulfa M., Hartanto R., Permanasari A. Caching strategy for Web application – a systematic literature review. *International Journal of Web Information Systems.* 2020. Vol. 16. No. 5. P. 545-569. DOI: https://doi.org/10.1108/IJWIS-06-2020-0032 (date of access: 17.10.2024)

5. Zahed M. Ishtiaque A., et al. A review on green caching strategies for next generation communication networks. *IEEE Access*. 2020. Vol. 8. P. 212709-212737. DOI: https://ieeexplore.ieee.org/abstract/document/9272291 (date of access: 17.10.2024)

6. Naeem M., Rehman M., Ullah R., Kim B. A Comparative Performance Analysis of Popularity-Based Caching Strategies in Named Data Networking. *IEEE Access.* 2020. Vol. 8. P. 50057-50077. DOI: 10.1109/ACCESS.2020.2980385 (date of access: 17.10.2024)

7. Naeem M., et al. Compound popular content caching strategy in named data networking. *Electronics.* 2019. Vol. 8. No. 7. P. 771. DOI: https://doi.org/10.3390/electronics8070771 (date of access: 17.10.2024)

8. Zhao L., Li H., Lin N., Lin M., Fan C., Shi J. Intelligent Content Caching Strategy in Autonomous Driving Toward 6G. *IEEE Transactions on Intelligent Transportation Systems.* 2022. Vol. 23. No. 7. P. 9786-9796. DOI: 10.1109/TITS.2021.3114199 (date of access: 17.10.2024)

9. Shuja J., et al. Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey. *Journal of Network and Computer Applications.* 2021. Vol. 181. P. 103005. DOI: https://doi.org/10.1016/j.jnca.2021.103005 (date of access: 17.10.2024)

10. Chen C., Jiang J., Fu R., Chen L., Li C., Wan S. An Intelligent Caching Strategy Considering Time-Space Characteristics in Vehicular Named Data Networks. *IEEE Transactions on Intelligent Transportation Systems.* 2022. Vol. 23. No. 10. P. 19655-19667. DOI: 10.1109/TITS.2021.3128012 (date of access: 17.10.2024)

11. Beckmann A., Bollmann M., Buchholz T., Geiser R., Kerpen D., Conrad J. Development of a Digital Collaborative Whiteboard. In: Stephanidis C., Antona M., Ntoa S. (eds) HCI International 2021 - Late Breaking Posters. HCII 2021. Communications in Computer and Information Science, vol. 1499. Springer, Cham. 2021. DOI: https://doi.org/10.1007/978-3-030-90179-0_31 (date of access: 17.10.2024)

12. Kušević T., Blažević D., Keser T. Comparison Functionalities of HTTP and MQTT Protocols. In: 31st International Conference on Organization and Technology of Maintenance (OTO, 2022). OTO 2022. Lecture Notes in Networks and Systems. Vol. 592. Springer, Cham. 2023. DOI: https://doi.org/10.1007/978-3-031-21429-5_5 (date of access: 17.10.2024)

13. Li C. et al. Collaborative caching strategy based on optimization of latency and energy consumption in MEC. *Knowledge-Based Systems.* 2021. Vol. 233. P. 107523. DOI: https://doi.org/10.1016/j.knosys.2021.107523 (date of access: 17.10.2024)

14. Кешування в NestJS, офіційна документація. URL: https://docs.nestjs.com/techniques/caching (дата звернення: 17.10.2024).

15. Кешування в Node.js, офіційна документація. URL: https://nodejs.org/docs/v20.17.0/api/modules.html#caching (дата звернення: 17.10.2024).

16. How to use Redis for Query Caching. URL: https://redis.io/learn/howtos/solutions/microservices/caching (дата звернення: 17.10.2024).

17. Офіційна документація npm пакету cache-manager. URL: https://www.npmjs.com/package/cache-manager (дата звернення: 17.10.2024).

18. Getting Started with Monitors, Datadog Docs. URL: https://docs.datadoghq.com/getting_started/monitors/ (дата звернення: 17.10.2024).

19. Набір інструментів для побудови API на основі NestJS. URL: https://docs.nestjs.com/controllers (дата звернення: 17.10.2024).

20. Керівництво по використанню Apache JMeter для тестування навантаження. JMeter User Manual. URL: https://jmeter.apache.org/usermanual/index.html (дата звернення: 17.10.2024).

21. Покроковий гайд по використанню Apache JMeter для тестування навантаження. Apache JMeter Distributed Testing Step-by-step. URL: https://jmeter.apache.org/usermanual/jmeter_distributed_testing_step_by_step.html (дата звернення: 17.10.2024).

22. Node.js Caching and Database Optimization for High-Performance APIs. URL: https://medium.com/@techsuneel99/node-js-caching-and-database-optimization-for-high-performance-apis-219f5280923b (дата звернення: 17.10.2024).

23. Leveraging Redis Cache: Optimizing Database Costs and Enhancing Application Performance. URL: https://medium.com/cloud-native-daily/leveraging-redis-cache-optimizing-database-costs-and-enhancing-application-performance-64b1df337530 (дата звернення: 17.10.2024).

24. Abolhassani B. et al. Fresh caching for dynamic content. IEEE INFOCOM 2021-IEEE Conference on Computer Communications. 2021. P. 1–10. URL: https://ieeexplore.ieee.org/abstract/document/9488731 (date of access: 17.10.2024)

25. Huang, X., Song, F., Ye, Y., Yang, X., Li, X. Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments. *IEEE Transactions on Mobile Computing.* 2020. Vol. 19(4). P. 852–864. DOI: 10.1109/TMC.2019.2902090 (date of access: 17.10.2024)

Borovskova Ye.A. DETERMINING OPTIMAL CACHING STRATEGIES FOR IMPROVING PERFORMANCE OF SERVER APPLICATIONS ON NESTJS

The article is dedicated to studying the performance of server applications in modern information systems that process large volumes of data in real time. The need to enhance performance for maintaining the stability of these applications under high-load conditions is emphasized, as it is crucial for resource optimization and reducing query processing time. It has been established that cache management is one of the most effective strategies, which reduces database access frequency, decreases CPU load, and shortens query processing time.

The article also highlights modern caching approaches and assesses their impact on the performance of server systems based on the NestJS framework. The research process included load testing methods, such as Apache JMeter, to simulate high-load environments, along with monitoring tools like Datadog for real-time caching efficiency assessment. Findings show that implementing caching reduces the average query processing time by 90%, halves CPU load, and decreases database access by 95%, thereby improving application performance and reducing infrastructure costs, particularly by lowering RAM and CPU resource consumption.

The importance of selecting appropriate caching strategies based on request specifics and available system resources is also defined. The efficiency of adaptive caching based on real-time analytics and TTL-based caching, which enables dynamic system adjustment for optimal performance, is highlighted. The need for careful analysis and configuration is stressed to avoid issues with stale data and ensure data consistency.

The article also analyzes the impact of selected caching strategies on infrastructure cost optimization. Results confirm that implementing modern cache management approaches significantly enhances the performance and stability of server applications under high-load conditions. Future research prospects include exploring machine learning integration for automated cache management, which could further optimize resource utilization based on historical data analysis and system load forecasting.

Key words: *performance optimization, server applications, cache management, data processing efficiency, storage strategies.*