DOI: https://doi.org/10.32347/2707-501x.2023.52(3).227-236
УДК 004.6

Б. Ю. Волох
аспірант кафедри інформаційних технологій проектування та прикладної математики,
https://orcid.org/0000-0003-2846-2621
Київський національний університет будівництва і архітектури, Київ

ІНТЕЛЕКТУАЛЬНЕ КЕШУВАННЯ У ВИСОКОНАВАНТАЖЕНИХ СИСТЕМАХ ЯК ВІДПОВІДЬ НА ОБМЕЖЕННЯ КЛАСИЧНИХ МЕТОДІВ КЕРУВАННЯ КЕШЕМ

Актуальність роботи обґрунтована важливістю оптимальності обробки даних у високонавантажених системах, що досягається шляхом застосування технології кешування. Досліджено проблематику ефективного управління потоками даних в сучасних інформаційних системах. Розглянуто основні функції і основні види кешування у високонавантажених інформаційних системах. З'ясовано роль кешування в забезпеченні масштабованості і стабільності високонавантажених систем в умовах зростаючих вимог до їхньої надійності, адаптивності, стабільності та продуктивності. Зазначено, що у реаліях сучасних динамічних, високонавантажених розподілених систем виникає все більше ситуацій, у яких ці методи стають недостатніми або неефективними. При цьому підвищення ефективності доступу до даних, балансування навантаження, зменшення затримок і забезпечення стійкості системи напряму залежить від правильно організованого механізму кешування Проаналізовано типові обмеження класичних методів керування кешем. Показано, що такі методи, як Least Recently Used, Least Frequently Used, First In First Out та їх модифікації, дедалі частіше демонструють обмеження в динамічному середовищі, де важливе значення відіграють зміни в структурі запитів, контекст об'єктів і варіативність поведінки користувачів. Як відповідь на обмеження класичних методів керування кешем для задач, де суттєво враховувати особливості запитів і поведінки користувачів різних груп, контекст і зв'язки між об'єктами, а так динаміку запитів на основі їх історії, обґрунтовано доцільність застосування інтелектуальних стратегій керування кешем. Показано перспективу розробки і впровадження в роботу високонавантажених систем інтелектуальних компонентів керування кешем дозволить їм навчатися на реальних даних, прогнозувати майбутні запити і приймати гнучкі рішення щодо збереження чи видалення об'єктів з кешу. Подальші дослідження вирішено спрямувати на розробку інтелектуального методу кешування, що здатен забезпечити ефективний доступу до актуально важливих даних в інформаційній інфраструктурі систем галузі архітектури і будівництва.

Ключові слова: адаптивність, база даних, дані, інформаційна система, оптимізація обробки запитів, сховище даних, технологія кешування, управління потоками даних.

Вступ. У сучасних інформаційних системах спостерігається стрімке зростання обсягів даних і кількості одночасних користувачів, що створює нові виклики для забезпечення стабільної та швидкої роботи сервісів. За оцінками Cisco Annual Internet Report [1], до 2022 року кількість пристроїв, підключених до мережі, перевищила 29 мільярдів, що суттєво збільшило навантаження на архітектури з доступом до розподілених даних. У таких умовах кешування є одним з ключових механізмів, що дозволяє зменшити навантаження на основне сховище даних системи, прискорити відповіді і покращити користувацький досвід.

Огляд актуальних наукових досліджень. Кешування – технологія, що використовується в інформаційних систем для підвищення продуктивності та швидкодії за рахунок тимчасового зберігання даних у швидкому проміжному сховищі (кеші) [2].

Метод (політика) управління кешем – це формалізований механізм прийняття рішень щодо роботи з кеш-пам'яттю, який визначає рішення щодо:

– даних, які потрібно додати до кешу;
– даних, які потрібно видалити з кешу у разі його заповнення;
– часу оновлення вміст кешу.

Основна задача кешування полягає в мінімізації звернень до основного джерела даних при повторних запитах. Це зменшує навантаження на основну базу даних і пришвидшує обробку повторних запитів шляхом збереження результатів запитів або найчастіше використовуваних даних у кеші, який забезпечує швидший доступ до інформації.

Основні функції кешування у високонавантажених інформаційних системах включають [3]:

1. Зменшення навантаження на базу даних – кешування дозволяє уникати повторних звернень до сховища при частих запитах одних і тих самих даних.
2. Пришвидшення відповіді на запити – отримання даних з кешу відбувається в рази швидше, ніж з диска або віддаленого сервера.
3. Забезпечення стабільної роботи під час пікових навантажень – кеш виступає як буфер, що допомагає системі справлятися з високим трафіком.
4. Адаптацію до поведінки користувачів – зберігання персоналізованих або регіональних даних дозволяє системі адаптуватися до запитів в реальному часі.
5. Підтримку роботи при збоях основного сховища даних – кеш дозволяє тимчасово обробляти запити навіть за відсутності доступу до бази даних.

6. Зниження витрат у хмарній інфраструктурі – зменшення трафіку і кількості звернень до бази даних дозволяє оптимізувати використання ресурсів.

7. Оптимізацію обміну даними між сервісами – кеш зменшує затримки при взаємодії мікросервісів і мінімізує дублювання запитів.

8. Збереження результатів складних обчислень – попередньо обчислені або фільтровані дані не потребують повторного розрахунку.

Основними видами кешування є [3]:

1. Кешування за місцем розташування.

1.1. Клієнтське кешування, що підходить для статичних ресурсів, таких як зображення чи файли стилів, і зменшує затримки між клієнтом і сервером, коли дані зберігаються на пристрої користувача;

1.2. Серверне кешування, що дозволяє прискорити виконання запитів на стороні серверу і знизити навантаження на основну базу даних, коли дані зберігаються на сервері.

2. Кешування за типом реалізації [2].

2.1.Cache-Aside для читання (Read) підходить для сценаріїв із частими операціями читання та повторюваними запитами, забезпечуючи швидший доступ до часто запитуваних даних. При цьому додаток спочатку перевіряє чи потрібні дані є в кеші, тоді (якщо даних немає) додаток отримує їх з бази даних і додає в кеш, надалі ці дані використовуються при наступних запитах;

2.2.Cache-Aside для запису (Write) забезпечує контроль над тим які дані залишаються актуальними в кеші після оновлення основної бази при записі або оновленні даних. Для цього додаток спочатку змінює дані в базі даних, після чого оновлює їх в кеші чи видаляє їх з кешу (залежно від того чи потрібні ці дані в майбутньому).

2.3.Read-Through Cache менш гнучкий, ніж Cache-Aside, але цей додаток автоматизує процес наповнення кешу. Якщо даних в кеші немає, то самостійно завантажує їх із бази даних, зберігає для наступних звернень і повертає;

2.4.Write-Through Cache забезпечує узгодженість між кешем і базою, оскільки при оновленні даних у додатку вони одночасно записуються в кеш і базу даних. Проте через синхронний запис це може спричиняти затримки.

2.5.Write-Behind Cache з високою продуктивністю для запису, але з ризиком тимчасової неконсистентності дані спочатку записуються в кеш, а потім із певною затримкою оновлюються в базі даних.

3. Кешування за типом даних:

3.1.Результати обчислень, статистика або сесії користувачів;

3.2.Великі файли, такі як зображення і відео (використовування спеціалізованих кешів, Content Delivery Network).

4. Кешування за часом життя даних:

4.1.Постійне кешування, коли дані зберігаються до явного видалення;

4.2.Тимчасове кешування, коли дані автоматично видаляються після закінчення певного часу.

В умовах обмеженого обсягу кешу однією з ключових задач системи є визначення того, які саме дані мають залишатися в кеші, а які – видалятися при отриманні нових. Ця задача вирішується за допомогою керування кешем.

Класичні методи керування кешем [4 – 8] застосовуються в умовах обмежених обчислювальних ресурсів і стабільних шаблонів запитів упродовж десятиліть. Проте в сучасних динамічних умовах функціонування високонавантажених розподілених інформаційних систем виникає все більше ситуацій, у яких ці методи стають недостатніми або навіть неефективними. Зокрема у [9] підкреслюється, що в аналітичних середовищах, де велика кількість об'єктів є проміжними результатами обчислень, класичні методи часто зберігають непотрібні дані, витісняючи водночас важливі. Це знижує продуктивність навіть у добре налаштованих системах і обумовлює актуальність досліджень, спрямованих на розробку і вдосконалення методів управління кешем, що здатні забезпечити ефективний доступу до актуально важливих даних.

Метою статті є аналіз класичних методів управління кешем та пошуку шляхів їх вдосконалення у напрямку враховування поведінки користувачів, контексту запитів і зміни у структурі даних високонавантажених системах.

Виклад основного матеріалу. Наразі в більшості інформаційних систем найчастіше використовуються прості евристики, що ґрунтуються на припущеннях щодо поведінки користувачів або характеру запитів [2].

За результатами аналізу класичних методів управління кешем (табл. 1) визначені групи обмежень, що унеможливлюють використання класичних методів у нових умовах, а саме:

1. Відсутність реакції до змін в системі. Алгоритми на кшталт LRU та LFU працюють за фіксованими правилами й не адаптуються до змін у поведінці користувачів, рівнів навантаження чи зміні структури даних. Ці алгоритми не мають механізму самонавчання чи автоматичного пристосування до нових умов. У період пікових навантажень (при надзвичайній ситуації в певному регіоні) користувачі можуть масово запитувати дані про об'єкти в одному місті. Проте класичні методи не "переорієнтуються" на нові пріоритети, зберігаючи в кеші об'єкти, що більше не використовуються [4].
2. Ігнорування різниці між користувачами. Сучасні інформаційні системи, обслуговують різні групи користувачів, кожна з яких має власні запити, різні періоди активності та розуміння важливості інформації. В класичних методах усі користувачі вважаються однаковими, що призводить до кешування нерелевантних для певного користувача об'єктів і частих промахів кешу [5].
3. Ігнорування контексту об'єктів. Ні LFU, ні LRU не здатні розрізняти, які об'єкти мають критично важливе значення і працюють

лише з лічильниками чи часовими мітками, ігноруючи тип об'єкта, географічну прив'язку, його зв'язок з іншими даними чи роль у процесі прийняття рішень. Унаслідок цього в кеші можуть залишатися об'єкти, які втратили актуальність, тоді як справді важливі дані витісняються. LFU може залишити в кеші менш важливий об'єкт тому, що до нього стабільно часто зверталися раніше [6].

4. Обмеження у передбаченні майбутніх запитів. Класичні методи здатні реагувати на те, що вже сталося, тобто приймають рішення на основі минулої активності, не враховуючи, що багато систем потребують проактивних дій.

5. Неврахування тимчасових піків активності. Реальні інформаційні системи стикаються зі піками активності, що мають тимчасовий характер (після публічного звіту чи події в новинах). Кеш має адаптуватися до таких хвиль навантаження, тимчасово збільшуючи пріоритетність певних об'єктів. Класичні методи не розрізняють постійну популярність і короткочасні сплески, тому реагують із затримкою або взагалі не реагують [7].

6. Відсутність механізмів самонавчання. Основне обмеження класичних методів полягає у відсутності навчання на основі існуючої історії запитів. Вони не аналізують поведінку системи, не виявляють закономірностей, зав'язків між запитами різних користувачів і не оновлюють правила роботи. На відміну від інтелектуальних підходів, класичні алгоритми залишаються незмінними, навіть якщо ситуація повторюється багато разів [8].

Усі ці обмеження пояснюють непридатність класичних методів управління кешем для задач, де важливо враховувати: особливості поведінки різних груп користувачів; контекст і зв'язки між об'єктами; динаміку запитів на основі історії запитів.

Потреба в урахуванні зазначених обмежень надає підстави для розробки і впровадження інтелектуальних методів управляння кешем, які дозволяють:

– формувати рішення на основі багатьох ознак одночасно, зокрема типу об'єкта, його контексту, частоти доступу, пріоритету користувача;

– автоматично адаптуватися до змін у середовищі, змінюючи методи в реальному часі залежно від ситуації (пікових навантажень чи нових джерел запитів);

– прогнозувати запити на основі історичних патернів, виявляючи закономірності у зверненнях користувачів і передбачаючи запити на об'єкти.

У високонавантажених інформаційних системах, де обробляються до мільйонів запитів щодня, кешування не просто виконує роль оптимізації, а стає критичною умовою забезпечення стабільності і масштабованості системи. Підвищення ефективності доступу до даних, балансування навантаження, зменшення затримок і забезпечення стійкості системи напряму залежить від правильно організованого механізму кешування.

Одним із ключових викликів у таких системах є баланс між швидкістю доступу до даних і узгодженістю, оскільки при великій кількості одночасних читань одна й та сама інформація може бути затребувана десятки разів протягом хвилини.

Таблиця 1

Основні класичні методи управління кешем

Метод	Принцип роботи	Переваги	Недоліки
LRU	Видаляє найдавніше використаний об'єкт	Проста реалізація; добре працює при локальності посилань	Не враховує частоту звернень; не адаптується до змін
LFU	Видаляє об'єкт із найменшою частотою звернень	Враховує історію використання; корисний для стабільного потоку запитів	Утримує «застарілі» популярні об'єкти; складна реалізація при великій кількості об'єктів
FIFO	Видаляє об'єкти у порядку їх надходження	Проста реалізація	Ігнорує корисність та частоту; витісняє важливі об'єкти
MRU	Видаляє об'єкт, що використовувався останнім	Ефективний при одноразових запитах	У більшості менш ефективний, ніж LRU
Random Replacement	Видаляє випадковий об'єкт незалежно від частоти чи часу	Проста реалізація; непогано працює у хаотичних запитах	Нестабільна ефективність; можлива втрата важливих об'єктів
2Q	Використовує дві черги для об'єктів з одноразовим і повторним доступом	Зменшує «забруднення» кешу; адаптується до короткострокових і довгострокових шаблонів	Ускладнена реалізація.
ARC	Балансує між LRU і LFU відповідно до поточного навантаження	Адаптивність до зміни шаблонів; не потребує ручного налаштування	Висока складність реалізації.
CLOCK	Імітує LRU з меншими витратами,	Економний по пам'яті; ефективний компроміс між простотою і точністю	Менш точний, ніж LRU; вимагає спеціальної структури для реалізації.
TTL	Кожному об'єкту призначається термін життя, при завершенні якого він видаляється автоматично	Для кешування тимчасових даних; простий у налаштуванні	Не враховує реальну потребу в об'єкті; можливі передчасні видалення

У таких випадках кешування дозволяє уникнути надмірних запитів до основної бази даних, що істотно знижує навантаження на неї та підвищує загальну пропускну здатність системи. Окрім цього кешування відіграє

ключову роль у зменшенні затримки відповіді. Час доступу до даних із пам'яті або локального кешу набагато менший, ніж з бази даних, особливо якщо вона розміщена на іншому сервері або у хмарній інфраструктурі. Це дуже важливо для інтерфейсів, які потребують миттєвого реагування, або сервісів, які забезпечують доступ до великих масивів аналітичної інформації.

Критичним аспектом високонавантажених систем є забезпечення стійкості в періоди пікових навантажень коли кеш дозволяє витримати навантаження без значної втрати продуктивності [10]. У протилежному випадку, коли кешування відсутнє або реалізоване неефективно, система ризикує втратити стабільність або припинити обслуговування запитів. Важливим є питання відмовостійкості при тимчасовій недоступності основного сховища даних, коли система може повернути останні збережені в кеші дані. Це дозволяє уникнути повної недоступності сервісу, що критично важливо для державних або кризових систем, де безперервність доступу є умовою ефективності роботи.

Також кешування дозволяє реалізовувати інтелектуальні сценарії персоналізації коли часто повторювані запити конкретних користувачів або регіональні дані можуть бути передчасно збережені в кеші для швидшого доступу до них пізніше. Таким чином, кешування стає не лише засобом підвищення швидкодії системи, а й інструментом її адаптації до реальних потреб користувачів.

У системах із мікросервісною або хмарною архітектурою дедалі ширше використовується розподілене кешування на базі Redis Cluster або Memcached, що дозволяє тримати кеш паралельно на кількох вузлах, синхронізувати дані, зберігати стійкість до збоїв окремих компонентів і забезпечувати горизонтальне масштабування. Водночас у роботах з машинного навчання для кешування почала формуватися нова парадигма – стратегія, яка не ґрунтується лише на частоті чи давності запитів, а враховує багатовимірний контекст.

У [11] показано здатність моделей машинного навчання враховувати специфіку користувацької поведінки, поточне навантаження, тип об'єктів, що кешуються. Це дозволяє моделі приймати кращі рішення щодо збереження і видалення даних, особливо в умовах нестабільного середовища.

У [12] запропоновано комбінований підхід, у якому модуль машинного навчання активується лише у випадках, коли стандартна евристика дає низьку точність. Такий гібридний підхід дозволяє досягти високої ефективності без значного зростання обчислювальних витрат, що робить його придатним для практичного використання в масштабованих системах.

Подальші дослідження планується присвятити розробці методу керування кешем, що базуються на машинному навчанні і здатен забезпечити ефективний доступу до актуально важливих даних в інформаційній інфраструктурі систем галузі архітектури і будівництва.

Висновки. Проведений аналіз класичних методів управління кешем показав, що вони дедалі частіше демонструють не достатню ефективність у динамічному середовищі, де важливо враховувати контекст запитів, поведінку користувачів і зміни у структурі даних. Зростаюча складність запитів і вимог до персоналізації призводить до того, що традиційні евристики уже не здатні забезпечити потрібний рівень адаптивності. У відповідь на це виникає потреба в розробці інтелектуальних методів, зокрема у машинного навчання, до управління кешем дозволить системам навчатися на реальних даних, прогнозувати майбутні запити і приймати гнучкі рішення щодо збереження чи видалення об'єктів з кешу.

Список використаної літератури

1. Cisco Annual Internet Report (2018–2022). URL: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html

2. Hazelcast. *What Is Caching? How Caching Works and Its Limitations*. URL: https://hazelcast.com/foundations/caching/caching/

3. Prisma. Database caching: Overview, types, strategies and their benefits. URL: https://www.prisma.io/dataguide/managing-databases/introduction-database-caching

4. Megiddo N., Modha D. *ARC: A Self-Tuning, Low Overhead Replacement Cache*, USENIX Conference on File and Storage Technologies (FAST), 2003. URL: https://www.usenix.org/legacy/event/fast03/tech/full_papers/megiddo/megiddo.pdf

5. Rotem-Gal-Oz A. *Fallacies of Distributed Computing Explained*, 2006. URL: https://arnon.me/wp-content/uploads/Files/fallacies.pdf

6. Swain D., Paikaray B., Swain D. AWRP: Adaptive Weight Ranking Policy for Improving Cache Performance, *Journal of Computing*, Vol. 3 (2), 2011. DOI: 10.48550/arXiv.1107.4851

7. Scully T., Jiang S., Zhang X. Learning-based Cache Replacement for High Performance and Dynamic Workloads. *Proceedings of the 2020 IEEE International Conference on Big Data*. 2020, p. 4042-4051.

8. M Liu E., Hashemi M., Swersky K., Ranganathan P., Ahn J. An Imitation Learning Approach for Cache Replacement, *International Conference on Machine Learning (ICML)*, 2020. DOI: https://doi.org/10.48550/arXiv.2006.16239

9. Gartner. Magic Quadrant for Distributed Data Management Solutions. URL: https://www.gartner.com/en/documents/4000163

10. CacheFly. *The Power of Machine Learning for Advanced CDN Caching Strategies*. 2022, URL: https://www.cachefly.com/news/the-power-of-machine-learning-for-advanced-cdn-caching-strategies

11. Cisco. *Global Networking Trends Report*, 2022. URL: https://storage.eventcheckin.co.kr/cisco/2023/CXO_symposium/data/2022_Global_Networking_Trend_Report_eng.pdf

12. Weiner M., Xu Y., "Performance Metrics for Distributed Systems", *ACM Computing Surveys*, 2022, Vol. 54, No. 7, P. 1–29.

References

1. Cisco. (2018–2022). *Cisco Annual Internet Report* URL: https://www.cisco.com/c/en/us/solutions/executive-perspectives/annual-internet-report/index.html

2. Hazelcast. *What Is Caching? How Caching Works and Its Limitations*. URL: https://hazelcast.com/foundations/caching/caching/

3. Prisma. *Database caching: Overview, types, strategies, and their benefits*. URL: https://www.prisma.io/dataguide/managing-databases/introduction-database-caching

4. Megiddo, N., & Modha, D. (2003). ARC: A Self-Tuning, Low Overhead Replacement Cache. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*. URL: https://www.usenix.org/legacy/event/fast03/tech/full_papers/megiddo/megiddo.pdf

5. Rotem-Gal-Oz, A. (2006). *Fallacies of Distributed Computing Explained*. URL: https://arnon.me/wp-content/uploads/Files/fallacies.pdf

6. Swain, D., Paikaray, B., & Swain, D. (2011). AWRP: Adaptive Weight Ranking Policy for Improving Cache Performance. *Journal of Computing*, *3*(2). URL: https://doi.org/10.48550/arXiv.1107.4851

7. Scully, T., Jiang, S., & Zhang, X. (2020). Learning-based Cache Replacement for High Performance and Dynamic Workloads. In *Proceedings of the 2020 IEEE International Conference on Big Data*, 4042–4051.

8. Liu, E. M., Hashemi, M., Swersky, K., Ranganathan, P., & Ahn, J. (2020). An Imitation Learning Approach for Cache Replacement. *International Conference on Machine Learning (ICML)*, URL: https://doi.org/10.48550/arXiv.2006.16239

9. Gartner. *Magic Quadrant for Distributed Data Management Solutions*. URL: https://www.gartner.com/en/documents/4000163

10. CacheFly. (2022). *The Power of Machine Learning for Advanced CDN Caching Strategies*. URL: https://www.cachefly.com/news/the-power-of-machine-learning-for-advanced-cdn-caching-strategies

11. Cisco. (2022). *Global Networking Trends Report*. URL: https://storage.eventcheckin.co.kr/cisco/2023/CXO_symposium/data/2022_Global_Networking_Trend_Report_eng.pdf

12. Weiner, M., & Xu, Y. (2022). Performance Metrics for Distributed Systems. *ACM Computing Surveys, 54*(7), 1–29.

Volokh Bohdan

Intelligent caching in high-load systems as a response to the limitations of classical cache management methods

The relevance of the work is substantiated by the importance of optimal data processing in highly loaded systems, which is achieved by using caching

technology. The problems of effective management of data flows in modern information systems are researched. The basic functions and main types of caching in high-load information systems are considered. The role of caching in ensuring the scalability and stability of highly loaded systems in the conditions of growing requirements for their reliability, adaptability, stability and performance is defined. It is specified that in the realities of modern dynamic, highly loaded distributed systems, there are more and more situations in which these methods become insufficient or ineffective. At the same time, increasing the efficiency of data access, load balancing, reducing latency and ensuring system stability directly depends on a properly organised caching mechanism. The typical limitations of classical cache management methods are analysed. It is shown that methods such as Least Recently Used, Least Frequently Used, First In First Out and their modifications are more often demonstrating limitations in a dynamic environment where changes in the structure of requests, the context of objects and variability of user behaviour play an important role. As a response to the limitations of classical cache management methods for tasks where it is essential to take into account the specifics of queries and user behaviour of different groups, the context and relationships between objects, as well as the dynamics of queries based on their history, the expediency of using intelligent cache management strategies is substantiated. The prospect of development and integration of intelligent cache management components into the operation of high-load systems is shown, which will allow them to learn from real data, predict future requests and make effective decisions on saving or deleting objects from the cache. It has been decided to focus further research on the development of an intelligent caching method that can provide efficient access to relevant data in the information infrastructure of systems in the field of architecture and construction.

Keywords: adaptability, data, data flow management, data storage, database, information system, query processing optimization, caching technology.

Посилання на статтю:

APA: Volokh Bohdan (2023). Intelligent caching in high-load systems as a response to the limitations of classical cache management methods. *Shliakhy pidvyshchennia efektyvnosti budivnytstva v umovakh formuvannia rynkovykh vidnosyn, 52(3), 227-236.*

ДСТУ: Волох Б. Ю. Інтелектуальне кешування у високонавантажених системах як відповідь на обмеження класичних методів керування кешем. *Шляхи підвищення ефективності будівництва*. 2023. № 52(3). С. 227-236.