DOI: https://doi.org/10.30837/IST-2024.P1.127

Аналіз алгоритмів кешування в Redis

Матвій Кучапін[1†], Марія Широкопетлєва[1*†] та Олена Шевченко[1†]

[1] *Харківський національний університет радіоелектроніки, проспект Науки, 14, Харків, 61166, Україна*

Abstract

Redis is a widely utilised database management system for various high-performance web applications, analytical tools, and data processing systems. Data caching is one of the principal methods to enhance the performance of such high-loaded systems. This paper analyses the existing approaches to cached data management in Redis, highlighting several common limitations associated with large and complex data structures, such as hashes, sets, lists, etc. It also outlines different data access patterns and the need to apply different caching strategies. The paper proposes the use of adaptive caching algorithms that take into account the data access patterns in other parts of complex structures to implement these strategies.

Ключові слова

Алгоритми кешування, Redis, LRU, LFU, TTL

1. Вступ

У сучасному світі інформаційних технологій, де обсяги даних постійно зростають, з'являються нові виклики щодо ефективної обробки та зберігання інформації. Великі обсяги даних вимагають від розробників програмних систем та баз даних забезпечення швидкого доступу до інформації, мінімізації затримок та оптимальне використання ресурсів.

Одним з ключових методів оптимізації роботи інформаційних систем є кешування, яке дозволяє знизити навантаження на основну базу даних, прискорити обробку запитів та зменшити час відгуку системи. Завдяки цьому даний підхід широко використовується у високонавантажених веб-додатках, системах аналітики та обробки даних, й у багатьох інших галузях, а для забезпечення кешування широко використовується Redis [1].

Метою роботи є дослідження механізмів управління кешованими даними у Redis, що впливають на ефективність використання пам'яті та швидкодію системи.

2. Обґрунтування вибору засобів кешування

Аналізуючи сучасні та популярні in-memory бази даних, що забезпечують швидкий доступ до даних та стабільну роботу систем під значним навантаженням [2], для дослідження було обрано Redis як основний інструмент для кешування. Це зумовлено його високою швидкодією та використанням оперативної пам'яті для зберігання даних, що забезпечує мінімальну затримку при обробці запитів. Redis є ідеальним рішенням для застосунків із критичними вимогами до швидкості обробки інформації [3]. Однак, незважаючи на всі переваги, управління складними структурами даних у Redis пов'язане

Information Systems and Technologies (IST-2024), November 26-28, 2024, Kharkiv, Ukraine

* Corresponding author.

† These authors contributed equally.

matvii.kuchapin@nure.ua (М. Кучапін); marija.shirokopetleva@nure.ua (М. Широкопетлєва); olena.shevchenko@nure.ua (О. Шевченко)

0009-0006-1953-2893 (М. Кучапін); 0000-0002-7472-6045 (М. Широкопетлєва); 0000-0003-3177-5530 (О. Шевченко)

з низкою труднощів [4]. Проблеми виникають під час роботи з такими структурами, як хеші, множини і списки, де окремі елементи відрізняються за важливістю та частотою використання. У таких випадках надмірне видалення все ще корисних даних або, навпаки, зберігання старих елементів, які споживають ресурси пам'яті, може негативно позначитися на продуктивності системи. Це призводить до зниження загальної продуктивності через необхідність відтворювати або завантажувати дані повторно. Водночас зберігання застарілої інформації призводить до нераціонального використання пам'яті та знижує ефективність кешування.

Основною проблемою в контексті складних структур даних є відсутність гнучких механізмів управління на рівні окремих елементів усередині цих структур. Це обмежує можливість оптимізації роботи над великими об'єктами, які можуть містити як рідка використовувані, так і часто запитувані елементи.

3. Аналіз існуючих підходів до управління даними в Redis

У сучасних системах управління кешованими даними ключову роль відіграють алгоритми видалення та оновлення даних. Redis, як одна з найпопулярніших систем кешування, реалізує декілька підходів для управління даними: TTL, LRU та LFU. Однак, кожен із цих методів має свої обмеження при роботі зі складними структурами, що впливає на ефективність та продуктивність системи.

TTL є стандартним механізмом керування терміном життя даних у Redis [5]. Він дозволяє призначати певний термін існування кожному ключу, після закінчення якого дані автоматично видаляються з кешу. Це забезпечує контроль над свіжістю даних та автоматичне очищення кешу від застарілих записів. Проте, у випадку великих структур TTL застосовується до всього об'єкта. Це означає, що після завершення терміну життя видаляється вся структура, навіть якщо частина елементів усе ще актуальна. В результаті виникає проблема надмірного видалення корисних даних. Відсутність гнучкого управління TTL на рівні окремих елементів у великих структурах обмежує можливості адаптації системи до різних рівнів важливості та частоти використання даних.

Алгоритм LRU видаляє найдавніше використані елементи, щоб звільнити місце під нові дані [6]. Це дозволяє зберігати у кеші найчастіше використовувані елементи. Однак, при роботі зі складними структурами Redis не завжди коректно визначає, які елементи потрібно видаляти. У випадках, коли деякі елементи у структурі використовуються рідко, але залишаються важливими, LRU може видалити їх, що призводить до втрати критично важливих даних і зниження ефективності кешування. Відсутність розмежування важливості елементів у великих структурах робить цей алгоритм менш ефективним для складних випадків.

LFU орієнтується на частоту використання даних і видаляє найменш часто запитувані елементи [6]. Він більш ефективний для сценаріїв, де частота використання є важливішим критерієм, ніж час останнього доступу. Проте LFU також має свої обмеження при роботі зі складними структурами, тому що він не враховує зв'язки між елементами у структурах, що може призводити до видалення важливих даних, якщо вони запитуються нечасто, але є критичними для програми. Крім того, цей алгоритм потребує додаткових ресурсів для відстеження частоти використання, що ускладнює його реалізацію та знижує продуктивність при високих навантаженнях.

Існуючі підходи управління кешем у Redis мають кілька спільних обмежень, зокрема:

- неефективне використання пам'яті при застосуванні TTL до великих структур даних. Весь об'єкт видаляється після закінчення терміну життя, незалежно від активності окремих елементів, що призводить до перевитрати оперативної пам'яті та необхідності повторного створення кешованих даних;

- відсутність можливості гнучкого управління TTL на рівні окремих елементів у великих структурах, що обмежує адаптивність системи до різних рівнів важливості та частоти використання даних. Це не дозволяє коректно видаляти застарілі елементи з великих структур, які можуть мати різну важливість;
- рідко використовувані елементи у великих структурах продовжують займати пам'ять навіть після того, як їх використання стає неактуальним.

В таблциці 1 наведено результати порівняння алгоритмів LRU та LFU.

Таблиця 5

Порівняльна таблиця алгоритмів LRU та LFU

Параметр	LRU	LFU
Критерій видалення	Час останнього доступу до ключа	Частота доступу до ключа
Переваги	Простота реалізації	Зберігає найпопулярніші ключі
Недоліки	Може видаляти часто використовувані, але нещодавно неактивні ключі	Повільна адаптація до змін у популярності
Підходить для	Динамічних патернів доступу	Стабільних патернів доступу

Дані характеристики вказують на те, що вибір між LFU та LRU залежить від конкретної системи та вимог до кешування. LRU є більш ефективним у системах з динамічними патернами доступу, де актуальність даних змінюється швидко. LFU, навпаки, підходить для систем зі стабільною популярністю даних, де певні ключі постійно користуються попитом. Різні патерни доступу до даних можуть впливати на ефективність кожного з цих алгоритмів, тому вибір алгоритму управління кешем повинен базуватися на аналізі поведінки користувачів та характеру даних у системі.

Загалом, використання стандартних механізмів управління кешем при роботі зі складними структурами даних може призвести до наступних негативних наслідків:

- старі та неактуальні дані можуть залишатися в пам'яті та займати місце, яке могло б бути використане для зберігання актуальної інформації;
- часте видалення та повторне створення великих структур даних збільшує навантаження на систему та призводить до збільшення часу відповіді;
- видалення актуальних даних через недосконалість політик витіснення може призвести до неконсистентності даних та негативно вплинути на користувацький досвід.

При зростанні обсягів даних та кількості користувачів обмеження пам'яті та недостатня гнучкість політик витіснення стають більш помітними, що може обмежити можливості систем.

Одним із перспективних напрямків є розробка динамічних механізмів TTL, які дозволять контролювати час життя даних на рівні окремих елементів складних структур, враховуючи їхню активність та важливість.

Крім того, сучасні дослідження орієнтуються на створення алгоритмів, що поєднують можливості LRU та LFU, але мають покращену гнучкість і здатність адаптуватися до змінних умов роботи системи [7]. Такий підхід дозволяє краще управляти пам'яттю, зберігаючи часто запитувані дані та своєчасно видаляючи рідко використовувані елементи, знижуючи загальну затримку доступу до кешованих даних.

Перспективи розвитку полягають у впровадженні комбінованих алгоритмів, які автоматично підлаштовуються під поточне навантаження та обсяг кешованої інформації

[8]. Подальша інтеграція таких адаптивних алгоритмів у кешуючі системи, зокрема Redis, дозволить ефективніше вирішувати проблеми масштабування кешованих даних, підвищувати продуктивність та забезпечувати високу швидкість доступу до даних у системах із високим навантаженням.

Одним із важливих напрямків у сучасних дослідженнях є адаптивні алгоритми кешування, які здатні пристосовуватися до динамічних патернів доступу до даних.

4. Висновки

Недосконаліть стандартних підходів також проявляється у неможливості тонкого налаштування під конкретні потреби додатків. Стандартні політики витіснення не враховують особливості доступу до даних у різних частинах складних структур. Це може обмежити ефективність кешування та призвести до нераціонального використання ресурсів.

Таким чином, стандартні підходи TTL, LRU та LFU мають суттєві недоліки при роботі зі складними структурами даних у високопродуктивних системах. Для подолання цих проблем необхідно розробити більш гнучкі та адаптивні механізми управління кешем, які враховують специфіку складних структур даних та характер доступу до них, забезпечуючи ефективне використання ресурсів та високу продуктивність системи, що є напрямком подальших досліджень.

Перелік посилань

[1] Redis Docs. 2024. URL: https://redis.io/docs/latest/ (дата звернення 12.10.2024).

[2] Top 27 In-Memory Databases Compared. 2024. URL: https://www.dragonflydb.io/guides/in-memory-databases.

[3] A. Kuzochkina, M. Shirokopetleva and Z. Dudar, Analyzing and Comparison of NoSQL DBMS, 2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), Kharkiv, Ukraine, 2018, pp. 560-564, doi: 10.1109/INFOCOMMST.2018.8632133.

[4] P. Zhang, et al, Redis++: A High Performance In-Memory Database Based on Segmented Memory Management and Two-Level Hash Index, 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, VIC, Australia, 2018, pp. 840-847, doi: 10.1109/BDCloud.2018.00125.

[5] J. Liu, X. Wan, Q. Zhu, T. Peng and X. Hu, Research on Adaptive Cache Mechanism Based on TTL, 2022 2nd International Conference on Networking, Communications and Information Technology (NetCIT), Manchester, United Kingdom, 2022, pp. 507-511, doi: 10.1109/NetCIT57419.2022.00125.

[6] Z. Li, D. Liu and H. Bi, CRFP: A Novel Adaptive Replacement Policy Combined the LRU and LFU Policies, 2008 IEEE 8th International Conference on Computer and Information Technology Workshops, Sydney, NSW, Australia, 2008, pp. 72-79, doi: 10.1109/CIT.2008.Workshops.22.

[7] W. Ma, Y. Zhu, S. Wang and Y. Bao, LearnedCache: A Locality-Aware Collaborative Data Caching by Learning Model, 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), Xiamen, China, 2019, pp. 718-726, doi: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00109.