

УДК 004.4

Кеширование и его реализация с использованием Redis в современных серверных приложениях

Мутовин Никита Сергеевич

n-mutovin2003@mail.ru

Борсук Наталья Александровна

borsuk@bmstu.ru

КФ МГТУ им. Н.Э. Баумана, Калуга, Россия

Рассмотрены принципы кеширования и роль Redis в повышении производительности серверных приложений. Описаны преимущества использования Redis как высокоскоростного хранилища данных в оперативной памяти для обработки задач с высокой нагрузкой, минимизации задержек и улучшения масштабируемости. Рассмотрены механизмы долговременного хранения данных (снимки RDB и журналы AOF), репликации для обеспечения высокой доступности, а также использование Redis Sentinel и Redis Cluster для отказоустойчивости и распределенной работы. Подчеркнута актуальность Redis в современных серверных приложениях, требующих высокой эффективности и надежности.

Ключевые слова: кеширование, Redis, серверные приложения, высокая доступность, отказоустойчивость, масштабируемость, оперативная память, долговременное хранение данных, RDB, AOF, Redis Sentinel, Redis Cluster, репликация, производительность

При решении поставленной задачи разработки сервиса для взаимодействия с подрядчиками необходимо реализовать приложение, обеспечивающее эффективное и надежное взаимодействие между заказчиком и подрядчиками.

Взаимодействие заключается в постановке задач различного рода со стороны заказчика, в поиске путей решения данных задач и их реализации с точки зрения исполнителя. Сервис должен позволять подрядчикам (исполнителям) получать задачи, оставлять отклики, предоставлять отчетность и участвовать в обсуждениях. Кроме того, необходимо интегрировать сервис с другими внутренними и внешними системами для обмена данными о задачах, статусах выполнения и сроках. Также важным аспектом является обработка данных в реальном времени, чтобы все участники могли своевременно получать актуальную информацию.

Кеширование в контексте такого сервиса необходимо по нескольким причинам. Во-первых, оно помогает снизить нагрузку на основной сервер базы данных, так как часто запрашиваемые данные могут храниться в быстром доступе, что ускоряет обработку запросов. Во-вторых, кеширование важно для обеспечения отказоустойчивости системы. Если основной сервер или база данных выходит из строя, кешированные данные могут быть использованы для предоставления резервной информации пользователям. Это позволяет сервису продолжать работу без значительных перебоев, используя актуальные данные, такие как статусы задач или список подрядчиков, которые могут быть извлечены из кеша.

Основная цель кеша — ускорение доступа к данным за счет их хранения на устройствах с высокой скоростью. Данные, сохраненные в кеше, доступны быстрее, чем те, что находятся на дисках, что особенно важно для данных, часто используемых в приложениях. Кеширование повышает производительность, снижает нагрузку на сервер и оптимизирует пропускную способность, а также улучшает взаимодействие с пользователем и повышает доступность системы даже при высокой нагрузке. Хранение данных в памяти также позволяет минимизировать запросы к базовым хранилищам и поддерживать скорость отклика [1].

Кеширование запросов к базе данных позволяет хранить результаты запросов временно, снижая нагрузку на сервер. Для поддержания актуальности кеша важно задавать политику удаления данных, используя срок действия и алгоритмы вытеснения, такие как LRU и LFU. Это позволяет определять, какие записи удаляются при переполнении кеша. Методы управления кешем включают установку срока действия и тегирование, что позволяет избирательно удалять устаревшие данные.

Системы кеширования, работающие в оперативной памяти, такие как Redis и Memcached, позволяют хранить часто запрашиваемые данные, включая результаты запросов к базе данных или ответы API, что значительно сокращает задержки при выполнении запросов. Хранение данных в оперативной памяти уменьшает необходимость повторных обращений к основным базам и увеличивает скорость обработки запросов [2].

Redis, ориентированный на скорость доступа и гибкость, представляет собой серверную базу данных, поддерживающую различные структуры данных, такие как строки, списки, множества и хеши, что делает его особенно удобным для временного хранения данных и потоковой обработки. Обычно Redis используется для кеширования данных, редко изменяемых, но часто запрашиваемых, например, сеансов пользователей и аналитических данных панелей управления. Благодаря поддержке высокого уровня доступности и плагинов Redis может использоваться как основная база данных в некоторых сценариях с высокой нагрузкой.

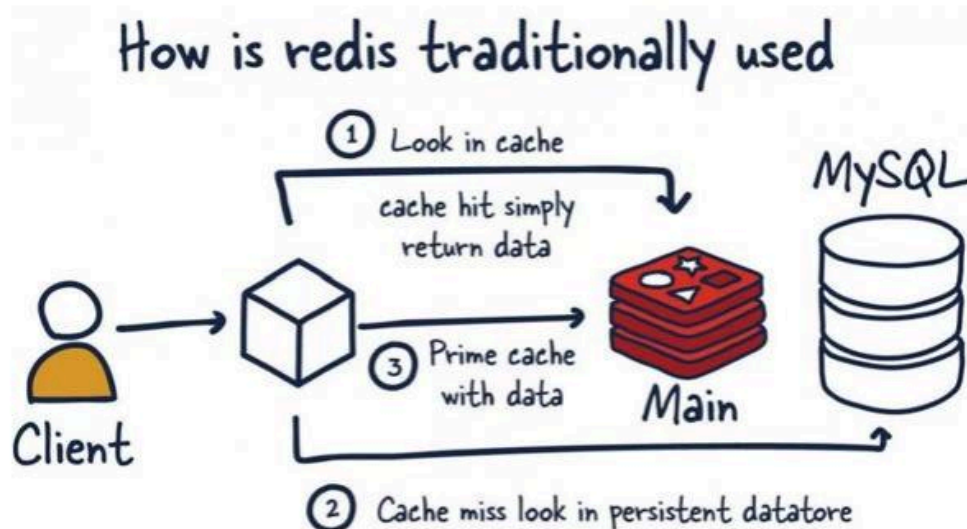
Redis эффективно сочетает функции кеша и хранилища данных, позволяя получать данные из памяти быстрее, чем с жестких дисков или SSD. Это делает его полезным решением для приложений, требующих высокой производительности и надежности хранения данных.

Изначально Redis чаще всего сравнивали с Memcached, с системой, в которой тогда не было и намека на долговременное хранение данных.

Главные различия между ним и Redis заключаются в том, что в Memcached имеется меньше типов данных, и в ограничениях, связанных с политикой вытеснения ключей. Memcached поддерживает лишь политику LRU (Least Recently Used), когда первыми вытесняются данные, которые не использовались дольше всех [3].

Еще одно отличие этих хранилищ заключается в том, что Redis — это однопоточная система, а Memcached — многопоточная. Memcached может

показывать отличные результаты производительности в ограниченных окружениях кеширования. А при использовании этой системы в распределенном кластере нужны дополнительные настройки. Redis же поддерживает подобные сценарии работы сразу после установки – см. рисунок [3].



Структура приложения с использованием Redis

Наиболее простой вариант использования Redis предполагает развертывание одного экземпляра системы, обеспечивающего доступное хранилище для ускорения работы приложений. Однако такой подход имеет недостатки: при сбое экземпляра все обращения к Redis становятся невозможными, что снижает производительность всей системы. При достаточном объеме памяти и серверных ресурсов экземпляр Redis может стать мощным решением для кеширования, обеспечивая значительный прирост производительности с минимальными затратами на настройку, даже если Redis размещен на той же машине, что и основное приложение.

Создание стабильной работы Redis требует понимания управления данными и механизмов хранения. Обработка запросов происходит из памяти, а при включенном постоянном хранении система создает резервные копии в формате RDB (снимки) или с использованием AOF (журнал команд), что поддерживает различные стратегии репликации и сложные топологии. При сбое системы и отсутствии активированного хранения данные теряются, в то время как включенное хранение позволяет восстановить их из RDB или AOF, обеспечивая готовность системы к новым запросам.

Конфигурация с ведущим и подчиненными узлами, где главный узел синхронизирует данные с подчиненными, повышает надежность и доступность за счет репликации. Главный узел имеет уникальный идентификатор и смещение репликации, что помогает отслеживать синхронизацию. В случае расхождения идентификаторов узлов возможна полная или частичная синхронизация данных. Если предыдущий идентификатор известен, частичная синхронизация позволяет минимизировать ресурсные затраты.

Redis Sentinel обеспечивает высокую доступность, мониторинг и автоматическое восстановление после сбоев, устраняя единую точку отказа с использованием кворума для принятия решений. Redis Cluster поддерживает горизонтальное масштабирование, распределяя нагрузку между узлами и сегментируя данные по хеш-слотам, что упрощает расширение и перераспределение данных.

При горизонтальном масштабировании нагрузка распределяется между множеством узлов, обеспечивая обработку больших объемов данных. Redis Cluster позволяет перераспределять только хеш-слоты при добавлении новых узлов, минимизируя влияние на доступность. Sentinel и Cluster делают Redis устойчивым и масштабируемым решением для высоконагруженных приложений, сочетая отказоустойчивость и производительность.

Таким образом, выше описывается целесообразность использования кэширования при реализации задачи создания сервиса для взаимодействия с подрядчиками различного рода работ. Это позволит обеспечить отказоустойчивость разрабатываемой системы и повысить производительность.

Литература

- [1] Таненбаум А.С., Стин М. *Распределенные системы: Принципы и парадигмы*. Москва, Питер, 2007.
- [2] Kleppmann M. *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*. Sebastopol, CA, O'Reilly Media, 2017.
- [3] Binstock A. *Caching: a primer*. San Francisco, CA, Practica Press, 2015.
- [4] Chen K., Li J. Cache Performance in Web Applications. *Journal of Web Engineering*, 2018, no. 17 (4), pp. 345–360.
- [5] Redis. IO. Documents. URL: <https://redis.io/docs/latest/> (дата обращения 14.11.2024).

Caching and its implementation using redis in modern server applications

Mutovin N.S. n-mutovin2003@mail.ru

Borsuk N.A. borsuk@bmstu.ru

BMSTU (KB), Kaluga, Russia

This article explores caching principles and the advantages of storing frequently accessed data for improved system efficiency. It discusses server-side caching with Redis and highlights its in-memory structure's benefits. Additionally, it covers Redis Sentinel and Redis Cluster's replication and high availability features, which enable resilience and scalability. Methods for persistent data storage in Redis, including RDB and AOF, are also reviewed, providing a balance between speed and reliability, making Redis suitable for various applications, from caching to distributed data storage.

Keywords: caching, Redis, server applications, in-memory storage, data persistence, high availability, replication, Redis Sentinel, Redis Cluster, scalability, fault tolerance, RDB snapshots, AOF logs, modern server architecture