Программные системы и вычислительные методы
Правильная ссылка на статью:
Золотухина Д.Ю. Эффективность распределённых кэширующих платформ в современных backend-архитектурах: сравнительный анализ Redis и Hazelcast // Программные системы и вычислительные методы. 2024. № 4. DOI: 10.7256/2454-0714.2024.4.72305 EDN: JNJVQQ URL: https://nbpublish.com/library_read_article.php?id=72305

Эффективность распределённых кэширующих платформ в современных backend-архитектурах: сравнительный анализ Redis и Hazelcast

Золотухина Дарья Юрьевна

независимый исследователь

394062, Россия, Воронежская область, г. Воронеж, пер. Антокольского, 4

dar.zolott@gmail.com

Статья из рубрики "Математическое и программное обеспечение новых информационных технологий"

DOI:

10.7256/2454-0714.2024.4.72305

EDN:

JNJVQQ

Дата направления статьи в редакцию:

11-11-2024

Аннотация: Объектом исследования являются две системы кэширования и распределенного хранения данных — Redis и Hazelcast, которые широко применяются для ускорения доступа к данным в высоконагруженных приложениях. В статье проводится всестороннее сравнительное исследование этих систем по ключевым аспектам, важным для эффективной работы с кэшированием: архитектурным особенностям, моделям управления памятью, подходам к кластеризации, механизмам отказоустойчивости и масштабируемости. Особое внимание уделяется исследованию возможностей работы с шаблонами кэширования и поддержке SQL-подобных запросов. Цель работы заключается в глубоком анализе преимуществ и ограничений Redis и Hazelcast в контексте кэширования данных, а также в выявлении их сильных и слабых сторон при различных нагрузках и сценариях эксплуатации. Методология исследования включает сравнительный анализ Redis и Hazelcast по ключевым аспектам, с последующим представлением результатов в виде сравнительной таблицы. Также было

проведено тестирование эффективности выполнения операций CRUD с использованием автоматизированных тестов, интегрированных в программу на платформе Spring Boot. Проведенное исследование показывает, что Redis, будучи однопоточной системой с быстрыми операциями записи и чтения, эффективен для простых и локализованных приложений, в то время как Hazelcast, поддерживающий многопоточность и динамическую кластеризацию, более эффективно справляется с большими объемами данных и распределенными задачами. Особым вкладом автора в исследование темы является комплексный сравнительный анализ этих систем с учетом их ключевых характеристик, таких как производительность, масштабируемость и отказоустойчивость, а также тестирование их работы в реальных сценариях. Новизна исследования заключается в детальном анализе применения Redis и Hazelcast для кэширования данных в высоконагруженных приложениях, что будет полезно для разработки и оптимизации инфраструктуры высокопроизводительных распределенных систем, которые требуют кэширования данных в реальном времени.

Ключевые слова:

кэширование, Redis, Hazelcast, производительность, CRUD операции, отказоустойчивость, многопоточность, кластеризация, in-memory хранение данных, распределенная система

Введение

С увеличением объёмов данных и требований к быстродействию систем кэширование становится ключевым механизмом, который позволяет ускорить доступ к данным, снизив нагрузку на основные базы данных. В современном мире распределённых систем важно иметь надёжное и производительное решение для кэширования, которое бы отвечало запросам как малых, так и крупных приложений. Среди множества доступных решений для кэширования два лидера рынка — Redis и Hazelcast — предлагают разнообразные возможности для обработки больших объёмов данных в оперативной памяти, однако различаются по архитектуре и подходам к распределённой работе с данными.

Обзор Redis и Hazelcast

Redis — это одна из ведущих in-memory систем для кэширования, которая позволяет значительно ускорить доступ к данным за счет их хранения в оперативной памяти. Это решение стало стандартом для множества приложений, требующих высокой скорости и производительности, так как Redis обеспечивает низкие задержки и высокую пропускную способность. Благодаря своим характеристикам, Redis применяется во множестве задач, от хранения пользовательских сессий до кэширования результатов сложных вычислений и временных данных, требующих мгновенного доступа [1].

Популярность Redis обусловлена его универсальностью и гибкостью в настройке, что делает его подходящим для высоконагруженных систем, которым важно быстро обрабатывать огромное количество запросов в реальном времени. В отличие от традиционных систем хранения данных, Redis ориентирован на мгновенный доступ к информации, что делает его идеальным для кэширования, где требуется поддерживать высокую производительность и скорость обработки данных. В сочетании с поддержкой отказоустойчивости и возможностей масштабирования, Redis продолжает оставаться одним из самых востребованных решений в области кэширования для современных

распределенных систем.

Hazelcast — это мощная распределенная система для кэширования и управления данными, которая ориентирована на обеспечение высокой производительности и гибкости в условиях крупных распределенных приложений. Как in-memory система, Hazelcast предназначена для того, чтобы ускорить доступ к данным за счет их хранения в оперативной памяти, что особенно важно для приложений, нуждающихся в быстрой обработке данных и низкой задержке. В отличие от многих традиционных решений, Hazelcast поддерживает многопоточную архитектуру и ориентирована на работу в кластерах, что позволяет ей обеспечивать высокую масштабируемость и надежность в условиях высокой нагрузки [2].

Благодаря возможности автоматического управления кластерами и распределения данных между узлами, Hazelcast активно используется для кэширования и поддержки временного хранения данных в крупных системах, где требуется балансировать нагрузку и минимизировать время отклика. Hazelcast также отличается встроенной поддержкой отказоустойчивости и высокой доступности, что делает его привлекательным для критически важных приложений. Эти качества позволяют Hazelcast уверенно занимать место среди лидеров решений для кэширования и in-memory хранения данных в современных корпоративных системах, обеспечивая как гибкость в настройке, так и надежность в эксплуатации.

Сравнительный анализ

В данной статье основное сравнение между Redis и Hazelcast проводится по семи ключевым категориям:

1. Многопоточность.

2. Шаблоны кэширования.

3. Кластеризация.

4. Построение запросов.

5. Память.

6. Вычисления.

7. Отказоустойчивость.

Redis — это однопоточная система, использующая высокопроизводительное ядро с минимальными затратами памяти. Такой подход позволяет запускать несколько экземпляров Redis на одной машине, распределяя нагрузку по ядрам ЦП и максимально используя ресурсы системы. Однопоточная модель обеспечивает простоту архитектуры и снижает вероятность возникновения проблемы рассинхронизации кластера — ситуации, когда изолированные части кластера работают независимо, что может приводить к различиям в данных между узлами при операциях записи [3]. Это особенно важно для обеспечения целостности данных.

В Hazelcast, напротив, используется многопоточная архитектура с пулом потоков для операций ввода/вывода. На каждом узле кластера операции ввода/вывода распределены между тремя типами потоков: потоки для приёма входящих запросов, потоки для чтения данных от других узлов и клиентов и потоки для записи данных. Благодаря этому Hazelcast может эффективно обрабатывать высокую нагрузку

ввода/вывода, масштабируясь в зависимости от потребностей. Однако многопоточная модель менее защищена от риска возникновения проблемы рассинхронизации кластера, что может вызывать сложности при обеспечении целостности в распределённом кластере [4].

Ключевое различие между Hazelcast и Redis при использовании их в качестве кэширующих решений заключается в гибкости подходов: Redis строго ориентирован на использование одного шаблона кэширования, тогда как Hazelcast поддерживает несколько различных схем. При применении Redis для кэширования данных, хранящихся в другом репозитории (например, в базе данных), требуется использовать шаблон cache-aside, что предполагает дополнительные сетевые переходы при каждом обращении к хранилищу.

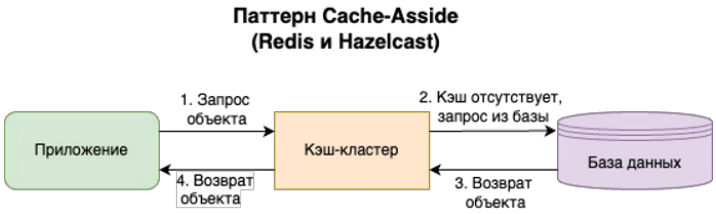

Рисунок 1. Паттерн cache-asside

Hazelcast поддерживает работу как с шаблоном cache-aside, так и с шаблонами read-through и write-through, при которых система автоматически взаимодействует с базой данных, выполняя чтение при отсутствии объекта в кэше или запись при изменении данных.

Рисунок 2. Паттерн read/write-through

Эти механизмы освобождают разработчика от необходимости вручную синхронизировать состояние между кэшем и основной базой данных, как это требуется в Redis, где также необходимо реализовывать логику для обновления и чтения данных из хранилища. В Hazelcast, напротив, вся логика взаимодействия с базой данных реализуется на уровне кэширующего слоя, что позволяет писать более чистый, структурированный и поддерживаемый код.

Redis и Hazelcast также имеют различия в реализации кластеризации — механизма объединения нескольких серверов или узлов в единую систему, что позволяет обеспечить высокую доступность, отказоустойчивость и масштабируемость приложений. В Redis кластеризация реализована через шардирование [5], что позволяет распределять данные между несколькими узлами. Каждый узел в этом случае хранит только часть

данных, а кластеры Redis автоматически определяют, какой узел отвечает за конкретный ключ. Тем не менее, процесс добавления или удаления узлов требует ручного перераспределения шардов, что увеличивает сложность управления. Кроме того, при выходе узла из строя необходимо вмешательство разработчиков для восстановления данных и функциональности, а операции с данными, расположенными на других узлах, сопряжены с дополнительными сетевыми затратами.

С другой стороны, Hazelcast изначально разработан как распределенная система и предлагает более совершенные функции кластеризации. Он обеспечивает автоматическое управление распределением данных и динамически перераспределяет их при добавлении или удалении узлов, что значительно упрощает администрирование кластера. При выходе узла из строя Hazelcast автоматически восстанавливает данные и перераспределяет нагрузку, что способствует повышению отказоустойчивости системы. Более того, узлы в Hazelcast могут взаимодействовать между собой без необходимости дополнительных сетевых обращений, что снижает задержки и улучшает производительность.

Таким образом, ключевые различия между Redis и Hazelcast в контексте кластеризации заключаются в том, что Redis требует ручного управления шардированием и восстановлением, тогда как Hazelcast автоматизирует эти процессы, обеспечивая более высокую отказоустойчивость и эффективность межузлового взаимодействия. Redis может быть более подходящим для простых сценариев, в то время как Hazelcast предпочтителен для распределенных систем, требующих высокой доступности и автоматического управления.

Запросы в Redis и Hazelcast, несмотря на общее использование модели «ключ/значение», имеют существенные различия в подходах к извлечению данных по свойствам значений, когда ключ неизвестен. Hazelcast предоставляет API предикатов, SQL-подобные выражения WHERE, а также механизмы проекций, что позволяет выполнять запросы к сложным объектам и JSON-структурам. Эти возможности делают Hazelcast более удобным для выполнения сложных выборок и анализа данных, сохраняя при этом гибкость работы с объектами различной структуры.

В Redis аналогичный подход требует иной архитектуры хранения: данные обычно представляются в виде хэша, где каждая строка сохраняется с ключом, сформированным на основе первичного ключа таблицы, и дополнительно включается в множество (Set) или отсортированное множество (Sorted Set). Такой метод хранения позволяет реализовать ограниченные возможности запросов, но требует дополнительных шагов при проектировании и может быть менее эффективным для сложных запросов по атрибутам, поскольку Redis не поддерживает полноценные предикаты и SQL-подобные запросы.

При сравнении Hazelcast и Redis в аспекте работы с памятью можно выделить несколько ключевых аспектов, которые отражают их разные архитектурные подходы и стратегии управления данными.

Redis ориентирован на хранение всех данных в оперативной памяти, что обеспечивает молниеносный доступ к ним. Его структура данных оптимизирована для минимизации использования памяти, и он поддерживает различные типы, такие как строки, хэши, списки и множества. Redis использует механизмы сжатия и эффективные аллокаторы памяти, что позволяет экономить ресурсы и увеличивать пропускную способность при работе с большими объемами данных. Однако такой подход подразумевает, что данные,

которые превышают доступный объем оперативной памяти, требуют использования дискового хранилища, что может снизить производительность.

С другой стороны, Hazelcast работает в среде Java и использует кучу памяти, управляемую сборщиком мусора. Это означает, что данные могут храниться как в оперативной памяти, так и в более стабильных хранилищах, таких как диски, что обеспечивает большую гибкость. Однако сборка мусора в Hazelcast может вызывать паузы в работе приложений, особенно при увеличении объема данных [6]. Это может привести к временным задержкам и снижению производительности, что критично для высоконагруженных систем.

Еще один важный аспект — распределенная природа хранения данных. Redis обычно работает как единый узел, хотя и поддерживает кластеризацию, которая делит данные между несколькими экземплярами, но при этом управление памятью может стать более сложным. Hazelcast, будучи изначально распределенной системой, обеспечивает автоматическое распределение данных по узлам, что позволяет лучше использовать память и увеличивать отказоустойчивость. Это позволяет Hazelcast справляться с увеличением объема данных, распределяя нагрузку между участниками кластера.

Redis обеспечивает максимальную скорость доступа к данным в памяти, тогда как Hazelcast предлагает большую гибкость и масштабируемость за счет своей распределенной архитектуры, но при этом может столкнуться с проблемами, связанными со сборкой мусора и управлением памятью.

Redis и Hazelcast оба позволяют выполнять вычислительные функции непосредственно на узле или участнике кластера, где хранится определенный набор данных, что особенно полезно в сценариях кэширования, требующих локального доступа к кэшированным данным. Например, в Hazelcast программу можно направить на узел, где находится конкретный ключ, что позволяет минимизировать сетевые задержки и ускорить доступ к данным, уже размещенным в кэше. В Redis также реализована схожая функциональность, благодаря которой можно эффективно обрабатывать кэшированные данные, избегая дополнительных вызовов к удаленным узлам.

Однако различия в подходах к распределенным вычислениям между Redis и Hazelcast также влияют на их кэширующие возможности. Hazelcast предоставляет Java-программе, запущенной на одном из узлов, доступ к данным на других узлах кластера, что позволяет использовать распределенный кэш более гибко и эффективно. Это особенно полезно при кэшировании данных, которые часто обращаются из разных частей приложения, поскольку Hazelcast поддерживает сценарии кэширования с межузловым взаимодействием и динамическим распределением данных. В Redis же кластеры и механизм Lua-скриптов не поддерживают доступ к кэшированным данным на других узлах, ограничивая кэширование локальной областью узла и снижая возможности гибкого управления кэшем.

Таким образом, Hazelcast предоставляет более мощную инфраструктуру для распределенных вычислений и кэширования, позволяя использовать кэш на нескольких узлах кластера как единую память, что оптимально для приложений с интенсивным межузловым взаимодействием и распределенными задачами. Redis, напротив, лучше подходит для задач, где кэширование и выполнение кода сосредоточены на конкретном узле без необходимости межузловой коммуникации, что делает его эффективным для локализованного кэширования и обработки данных в пределах одного узла.

Отказоустойчивость является важным аспектом в проектировании распределенных

систем, обеспечивающим непрерывность работы приложений даже в условиях сбоя отдельных компонентов [7]. В контексте Redis отказоустойчивость достигается через механизмы репликации и использование Sentinel для мониторинга состояния узлов. В данной архитектуре основной сервер (мастер) может иметь одну или несколько реплик, которые периодически обновляются для синхронизации данных. В случае выхода основного узла из строя, реплика может быть повышена до статуса мастера. Однако этот процесс требует ручного вмешательства или дополнительной настройки Sentinel, что может вызвать временные простои.

Hazelcast изначально разработан с акцентом на автоматизацию процессов отказоустойчивости. Каждый узел в кластере хранит реплики данных [8], что позволяет системе эффективно реагировать на сбои. При выходе узла из строя Hazelcast автоматически перенаправляет запросы к другим узлам, содержащим актуальные копии данных, что минимизирует время простоя и повышает доступность системы. Более того, механизм автоматического восстановления и перераспределения данных позволяет избежать необходимости ручного управления и снижает риски, связанные с потерей данных.

Таким образом, ключевое отличие в подходах к обеспечению отказоустойчивости между Redis и Hazelcast заключается в том, что Redis требует значительных усилий со стороны разработчиков для поддержания работоспособности системы, тогда как Hazelcast предлагает более надежное и автоматизированное решение. Это делает Hazelcast более предпочтительным вариантом для приложений, требующих высокой доступности и надежности в условиях сбоя отдельных компонентов.

Результаты сравнения кратко представлены в сравнительной таблице.

Таблица 1.

Сравнение Redis и Hazelcast

Аспект	**Redis**	**Hazelcast**
Потоки	Однопоточный	Многопоточный
Шаблоны кэширования	Cache-aside	Cache-aside, read-through, write-through
Тип кластеризации	Шардирование с ручным управлением	Автоматическое управление и перераспределение
SQL-подобные запросы	Не поддерживает	Поддерживает
Память	Быстрый доступ к данным, но ограничен объемом оперативной памяти	Потенциальные задержки из-за сборки мусора, но есть возможность обработки больших объемов данных
Вычисления	Ограничен работой с данными на одном узле	Распределенные вычисления с автоматической балансировкой нагрузки по узлам
Отказоустойчивость	При сбое требует	При сбое автоматически

	ручного вмешательства и настройки	перенаправляет запросы

Тестирование производительности Redis и Hazelcast для операций CRUD

Исходные данные

Программа, содержащая тесты, написана на платформе Java Spring Boot. Тесты проводятся на наборе данных, помещенных в файл формата CSV. Он включает в себя более 250 000 уникальных записей, каждая из которых представлена различными типами данных (строки, целые числа, длинные целые числа и числа с плавающей точкой).

Используемый клиент Java для Redis — Jedis. Jedis — это клиентская библиотека Java для взаимодействия с базой данных Redis, предоставляющая удобный и высокопроизводительный API для выполнения операций с данными в Redis. Jedis поддерживает основные команды Redis, а также расширенные функции [9].

Используемый клиент Java для Hazelcast — Hazelcast Predicates API. Он предоставляет интерфейс для выполнения запросов по заданным условиям в распределённых структурах данных, таких как IMap. С его помощью можно фильтровать и находить данные по сложным условиям, подобным SQL-запросам, что обеспечивает гибкость и высокую производительность при работе с распределёнными данными [10].

Набор проведенных тестов включал в себя:

- Создание (Create): исходные данные загружаются и вставляются в структуру HSET (Redis) и IMap (Hazelcast). Тест проводится в течение 1 часа. На протяжении этого времени данные читаются из CSV файла и добавляются в базу, пока не истечёт время теста.
- Чтение (Read): выполняется 5 типов запросов с разными параметрами, которые выбираются случайным образом из предварительно определённого множества. Подготавливается набор запросов, которые выполняются по очереди в течении 1 часа. Количество выполняемых запросов в секунду фиксируется.
- Обновление (Update): аналогично тесту чтения, но с другим набором запросов. Время выполнения составляет 1 час. Результат — количество обновлённых строк в секунду.
- Удаление (Delete): тест аналогичен обновлению, выполняется в течение 1 часа, фиксируется количество удалённых строк в секунду.

Конфигурация оборудования

Тесты проводились на машине со следующими характеристиками:

- Процессор: Apple M2 @ 3.49 GHz, 8 ядер, 8 потоков.
- Оперативная память: 16 ГБ.
- Накопитель: 256GB PCIe® NVMe™ M.2 SSD.

Результаты тестирования производительности

Create: Redis и Hazelcast показали практически аналогичные результаты при вставке данных, обрабатывая одинаковое количество запросов в секунду. Redis показал в среднем вставку 1040 записей в секунду, Hazelcast – 10020 записей.

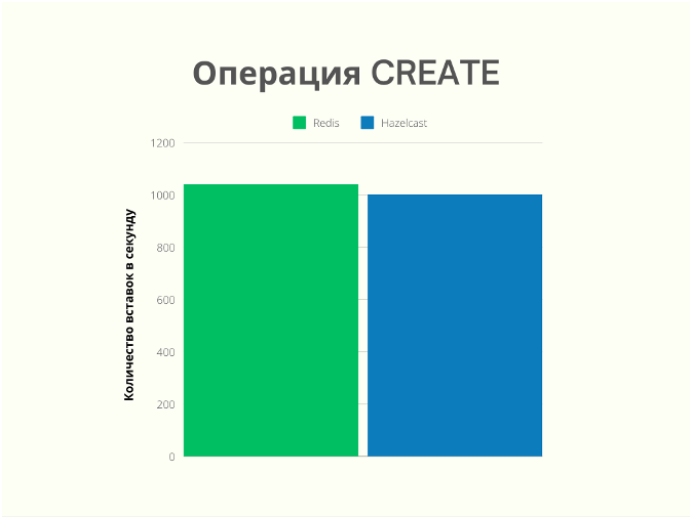

Рисунок 3. Результат операции Create

Read: Redis показал более высокую производительность при чтении данных. Это объясняется более высокой эффективностью библиотеки Redis (Jedis), что позволило быстрее обрабатывать запросы. Redis читал в среднем 14 записей в секунду, Hazelcast – 4.Однако Redis обеспечивает высокую скорость чтения за счет полного хранения данных в памяти, что делает его более требовательным к объему RAM. Таким образом, Redis понадобилось практически в два раза больше памяти, чем Hazelcast, для хранения того же объёма данных (Redis около 1300 байт на запись, Hazelcast около 650 байт).

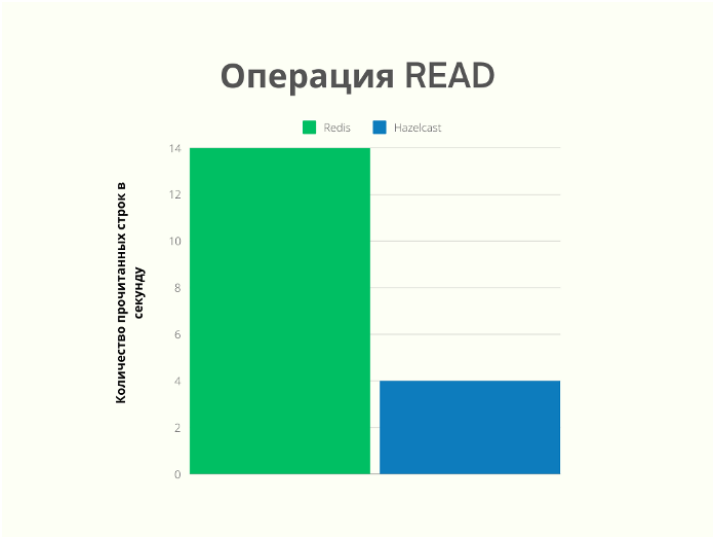

Рисунок 4. Результат операции Read

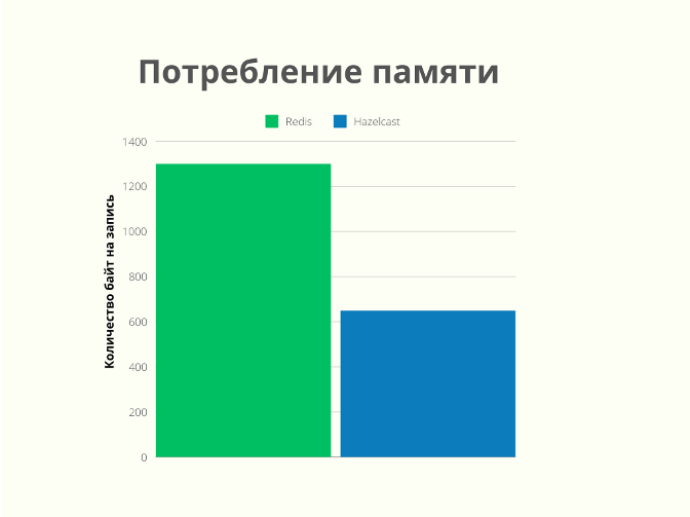

Рисунок 5. Потребление памяти

Update и Delete: Оба решения показали сопоставимую производительность при тестировании операций обновления и удаления данных, обеспечивая минимальные задержки и высокую скорость выполнения. Redis в среднем обновлял 1203 записи в секунду, Hazelcast – 1066 в секунду. При удалении Redis в среднем показал результат 1380 записей в секунду, Hazelcast – 1275 записей в секунду.

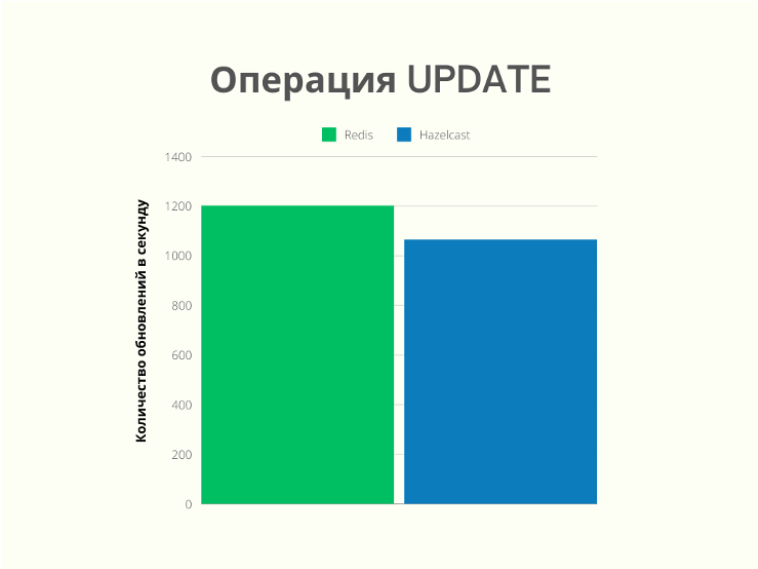

Рисунок 6. Результат операции Update

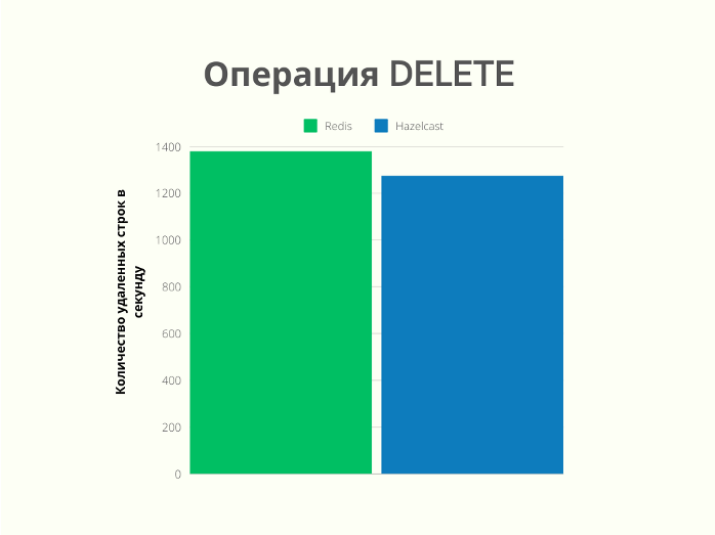

Рисунок 7. Результат операции Delete

Выводы

В ходе проведенного исследования был осуществлен сравнительный анализ двух популярных систем кэширования и распределенного хранения данных — Redis и Hazelcast. Оценка их архитектуры, производительности, механизмов отказоустойчивости и масштабируемости позволила выявить ключевые особенности, влияющие на выбор оптимального решения в контексте высоконагруженных распределенных приложений.

Redis, характеризующийся однопоточной архитектурой и высокой производительностью в операциях с данными, демонстрирует низкие задержки и быстрый доступ к данным, что делает его подходящим для приложений, ориентированных на быструю работу с данными в памяти. Однако его ограниченная гибкость в вопросах кластеризации и отказоустойчивости может потребовать дополнительных усилий при управлении несколькими узлами и обеспечении доступности данных в более сложных сценариях. Redis оптимален для горизонтально масштабируемых систем с простыми требованиями к распределению данных.

Система Hazelcast с многопоточной архитектурой и встроенной поддержкой кластеризации предоставляет более гибкие и масштабируемые возможности для построения распределенных систем. Механизмы автоматического распределения данных и масштабирования значительно упрощают администрирование и повышают отказоустойчивость. Hazelcast более эффективно справляется с интенсивными межузловыми взаимодействиями и высокими требованиями к отказоустойчивости, что делает его предпочтительным решением для крупных приложений с высокой нагрузкой.

Тестирование производительности показало, что Redis превосходит Hazelcast по скорости выполнения операций чтения. Обе системы показали схожие результаты при операциях обновления и удаления данных, но Redis требует большего объема оперативной памяти, что следует учитывать при выборе решения для ресурсоемких приложений.

Таким образом, выбор между Redis и Hazelcast должен основываться на специфике задач. Redis более подходит для приложений, где критична высокая производительность при работе с данными в памяти, а Hazelcast является более оптимальным вариантом для масштабируемых распределенных систем, требующих высокой отказоустойчивости и гибкости.

Библиография

1. Carlson J. Redis in Action. New York: Manning, 2013.
2. Johns M. Getting Started with Hazelcast. Birmingham: Packt Publishing, 2015.
3. Биллиг, В. А. Параллельные вычисления и многопоточное программирование. Москва: ИНТУИТ, 2016.
4. Кадомский А. А., Захаров В. А. Эффективность многопоточных приложений // Научный журнал. 2016. №7 (8). URL: https://scientificmagazine.ru/images/PDF/2016/8/Nauchnyj-zhurnal-7-8.pdf (дата обращения: 08.11.2024).
5. Бойченко А. В., Рогожин Д. К., Корнеев Д. Г. Алгоритм динамического масштабирования реляционных баз данных в облачных средах // Статистика и экономика. 2014. №6-2. URL: https://statecon.rea.ru/jour/article/view/584/566 (дата обращения: 08.11.2024).
6. Филатов А. Ю., Михеев В. В. Анализ эффективности потоково-локальной сборки мусора в распределённых системах хранения и обработки данных // Вестник СибГУТИ. 2022. №1 (57). URL: https://vestnik.sibsutis.ru/jour/article/view/122/126 (дата обращения: 08.11.2024).
7. Голева А. И., Стороженко Н. Р., Потапов В. И., Шафеева О. П. Математическое моделирование отказоустойчивости информационных систем // Вестник НГУ. Серия: Информационные технологии. 2019. №4. URL: https://intechngu.elpub.ru/jour/article/view/110/98 (дата обращения: 08.11.2024).
8. Борсук Н. А., Козеева О. О. Анализ методов репликации баз данных при разработке онлайн-сервиса // Символ науки. 2016. №11-3. URL: https://os-russia.com/SBORNIKI/SN-2016-11-3.pdf (дата обращения: 08.11.2024).
9. Intro to Jedis – the Java Redis – URL: https://www.baeldung.com/jedis-java-redis-client-library (date of access: 08.11.2024).
10. Predicates API – URL: https://docs.hazelcast.com/hazelcast/5.5/query/predicate-overview (date of access: 08.11.2024).

Результаты процедуры рецензирования статьи

В связи с политикой двойного слепого рецензирования личность рецензента не раскрывается.
Со списком рецензентов издательства можно ознакомиться здесь.

В статье рассматривается сравнительный анализ двух ведущих in-memory решений для кэширования в распределённых системах — Redis и Hazelcast. Оба инструмента используются для повышения производительности современных распределённых систем за счёт ускорения доступа к данным и снижения нагрузки на основные базы данных. Основное внимание уделено сравнительным характеристикам архитектуры, отказоустойчивости, масштабируемости и производительности, что позволяет оценить их применимость в контексте конкретных задач.
Методология исследования базируется на сравнительном анализе архитектурных особенностей Redis и Hazelcast, а также на тестировании производительности этих платформ при выполнении операций CRUD. Тестирование проводилось с использованием Java Spring Boot и клиентских библиотек, таких как Jedis и Hazelcast Predicates API. Результаты тестирования проанализированы для различных операций: создания, чтения, обновления и удаления данных, что позволило сделать выводы о производительности и потреблении ресурсов.
Актуальность исследования обусловлена ростом объемов данных и требованиями к быстродействию современных распределённых систем. В условиях необходимости

масштабирования приложений и увеличения скорости обработки запросов выбор оптимального кэширующего решения становится важным этапом проектирования системы. Поэтому анализ преимуществ и ограничений Redis и Hazelcast является своевременным и востребованным.

Научная новизна работы заключается в комплексном сравнительном анализе Redis и Hazelcast, проведённом в контексте их применения в современных backend-архитектурах. В статье подробно рассматриваются ключевые аспекты, влияющие на производительность и отказоустойчивость, такие как многопоточность, поддержка различных шаблонов кэширования, особенности кластеризации и работа с памятью. Также важным вкладом является предоставление результатов тестирования на реальных данных, что подтверждает выводы о практической применимости рассматриваемых решений.

Статья отличается хорошо структурированным изложением материала. Введение предоставляет необходимый контекст и объясняет значимость исследования. Основная часть разделена на тематические блоки, каждый из которых рассматривает отдельный аспект сравнения, что позволяет легко следовать ходу анализа. Результаты тестирования представлены с использованием наглядных иллюстраций и графиков, что способствует более глубокому пониманию полученных данных.

В статье сделаны важные выводы относительно преимуществ и недостатков Redis и Hazelcast в зависимости от задач, стоящих перед разработчиком. Redis демонстрирует высокую производительность в операциях чтения данных, однако требует большего объёма оперативной памяти и ручного управления при кластеризации. Hazelcast предлагает более гибкие возможности для распределённых систем, включая автоматическое масштабирование и управление отказоустойчивостью. Такие результаты будут интересны специалистам в области разработки высоконагруженных приложений и архитектуры распределённых систем, а также всем, кто рассматривает вопросы оптимизации производительности с использованием кэширующих решений.

Статья представляет собой ценный вклад в исследование эффективности распределённых кэширующих платформ. Проведённый анализ выявляет сильные и слабые стороны Redis и Hazelcast, что помогает сделать обоснованный выбор при проектировании backend-архитектуры современных приложений. Работа написана в академическом стиле, структура изложения логична и позволяет легко воспринимать материал. Рекомендуется к публикации, так как содержит ценные данные и выводы, которые могут быть полезны широкой аудитории специалистов и исследователей в области разработки программного обеспечения.