Столыпинский вестник

Научная статья

Original article

УДК 004.052

АНАЛИЗ СЕРВЕРНЫХ ВЫЧИСЛЕНИЙ

ANALYSIS OF SERVER COMPUTING

Горячкин Борис Сергеевич, кандидат технических наук, доцент; Московский государственный технический университет им. Н.Э. Баумана (105005, Москва, 2-я Бауманская ул., д. 5, с. 1), тел. 8(499) 263-63-91, bsgor@mail.ru

Зудин Алексей Максимович, магистрант; Московский государственный технический университет им. Н.Э. Баумана (105005, Москва, 2-я Бауманская ул., д. 5, с. 1), тел. 8(499) 263-63-91, alexeyz.zudin@yandex.ru

Boris S. Goryachkin, candidate of technical sciences, associate professor; Moscow State Technical University named after. N.E. Bauman (105005, Moscow, 2nd Baumanskaya st., 5, bldg. 1), tel. 8(499) 263-63-91, bsgor@mail.ru

Zudin A. Maksimovich, master's student; Moscow State Technical University named after. N.E. Bauman (105005, Moscow, 2nd Baumanskaya st., 5, bldg. 1), tel. 8(499) 263-63-91, alexeyz.zudin@yandex.ru

Аннотация. В условиях динамичного развития электронной коммерции и растущей конкуренции на рынке онлайн-торговли, эффективное управление информацией о товарах и оперативное обновление контента на витрине становятся ключевыми стратегическими задачами. Настоящая работа направлена на исследование подходов к решению проблемы неконсистентности

данных и оптимизации процессов обновления информации на витрине электронного маркетплейса. Результаты: в рамках исследования рассматриваются механизмы кэширования данных как инструмент оптимизации производительности витрины онлайн-торговли, с акцентом на сокращении важных параметров, таких как время ответа сервиса и время инвалидации кэша. Особое внимание уделяется методам инвалидации кэша, направленным на обеспечение высокой скорости обновления данных и минимизацию негативных последствий, таких как некорректное отображение товаров и потеря консистентности информации. Практическая значимость: результаты исследования позволят выявить преимущества и недостатки различных подходов к кэшированию данных, а также сформулировать рекомендации по оптимизации процессов управления информацией на витрине онлайн-торговых площадок, что важно для повышения их производительности и конкурентоспособности в условиях современного рынка электронной коммерции.

Abstract. In the context of the dynamic development of e-commerce and growing competition in the online trading market, effective management of product information and prompt updating of content on the showcase are becoming key strategic objectives. This work is aimed at researching approaches to solving the problem of data inconsistency and optimizing the processes of updating information on the showcase of the electronic marketplace. Result: the research examines data caching mechanisms as a tool to optimize the performance of an online retail storefront, with an emphasis on reducing important parameters such as the service response time and cache invalidation time. Particular attention is paid to cache invalidation methods aimed at ensuring high data update speed and minimizing negative consequences, such as incorrect display of goods and loss of consistency of information. Practical significance: the results of the study will reveal the advantages and disadvantages of various approaches to data caching, as well as formulate recommendations for optimizing information management processes on the showcase of online trading

platforms, which is important to increase their productivity and competitiveness in the modern e-commerce market.

Ключевые слова: *Кэширование данных, инвалидация кэша, оптимизация производительности, консистентность данных.*

Keywords: *Data caching, cache invalidation, performance optimization, data consistency.*

Введение

В условиях динамичного развития электронной коммерции и растущей конкуренции на рынке онлайн-торговли, эффективное управление информацией о товарах и оперативное обновление контента на витрине становятся ключевыми стратегическими задачами. В данной работе мы обратим внимание на инновационные подходы к решению проблемы неконсистентности данных и оптимизации процессов обновления информации на витрине электронного маркетплейса.

Целью данного исследования является рассмотрение механизмов кэширования данных как средства оптимизации производительности витрины онлайн-торговли, при этом уделяя особое внимание сокращению важных параметров, таких как время ответа сервиса и время инвалидации кэша.

Мы приступим к изучению методов, применяемых в современных информационных системах, сфокусировавшись на опыте и практике крупного маркетплейса, предоставляющего доступ к более чем ста сервисам и обслуживающего сотни тысяч запросов в секунду.

Основное внимание будет уделено подходам к инвалидации кэша, обеспечивающим не только высокую скорость обновления данных на витрине, но и минимизацию негативных последствий, таких как некорректное отображение товаров и потеря консистентности в информации. Мы рассмотрим эволюцию стратегий кэширования, начиная от простого ленивого кэширования и заканчивая инновационными методами инвалидации, основанными на событиях из брокера сообщений Kafka.

Проведенный анализ позволит выявить преимущества и недостатки различных подходов, а также сформулировать рекомендации по оптимизации кэширования данных для повышения производительности и конкурентоспособности маркетплейса. Все эти аспекты являются важными для понимания современных тенденций в области электронной коммерции и создания эффективных стратегий управления информацией на витрине онлайн-торговых площадок.

Обзор сервисной части

facade - сервис, который мы будем оптимизировать выступает в роли надежного фасада [1], обеспечивающего доступность товаров для всех ключевых сервисов витрины маркетплейса. Этот сервис является связующим звеном между каталогом, поиском, карточкой товара, корзиной, страницей оформления заказа, списком избранного, кабинетом продавца и другими функциональными блоками.

facade обслуживает более 100 клиентов, предоставляя им актуальную информацию о товарах. Запросы поступают из различных разделов сайта и мобильного приложения, охватывая все уголки, где присутствует информация о товарах. Каждый раздел сайта генерирует от одного до нескольких запросов к facade, добавляя динамизма обработке данных.

В период осенних распродаж, таких как День холостяка и Черная пятница, facade сталкивается с задачей удержания высокой нагрузки. Целью является обеспечение стабильной работы при 1 миллионе запросов в секунду (RPS). Важно отметить, что на текущих нагрузочных тестах сервис успешно справляется с 1.2 миллионами RPS, что подтверждает его готовность к экстремальным условиям.

На сегодняшний момент средняя дневная нагрузка на facade достигает впечатляющих 300 тыс. RPS. Это отражает постоянную активность пользователей, обращающихся к сервису для получения информации о товарах.

Пиковая нагрузка в 1.2 миллиона RPS на facade в период осенних распродаж – результат успешных нагрузочных тестов. Сервис

продемонстрировал не только способность к обработке такого объема запросов, но и резервные возможности для дополнительных вызовов.

facade, являющийся ключевым элементом нашей онлайн-платформы, представляет собой 666 инстансов, разработанных на языке программирования Golang [2]. Каждый инстанс оборудован 7 ядрами CPU и 7.5 Gb RAM, обеспечивая высокую производительность и отзывчивость сервиса.

Оптимизация времени ответа и эффективное использование ресурсов достигнуты благодаря интеграции 201 шард кэшей [3] с использованием memcached [4], общим объемом оперативной памяти 1.1 Tb. Этот мощный механизм кэширования способствует переиспользованию данных, что в конечном итоге сокращает время ответа сервиса.

facade в настоящее время кэширует информацию от 21 мастер-системы, включая готовые ответы нашего сервиса и сырые данные, используемые в реальном времени для расчетов. Этот обширный объем кэшируемых данных обеспечивает эффективное управление информацией о товарах и обеспечивает наших клиентов актуальной и полной информацией. Ниже на рисунке 1 представлена архитектура серверного взаимодействия.

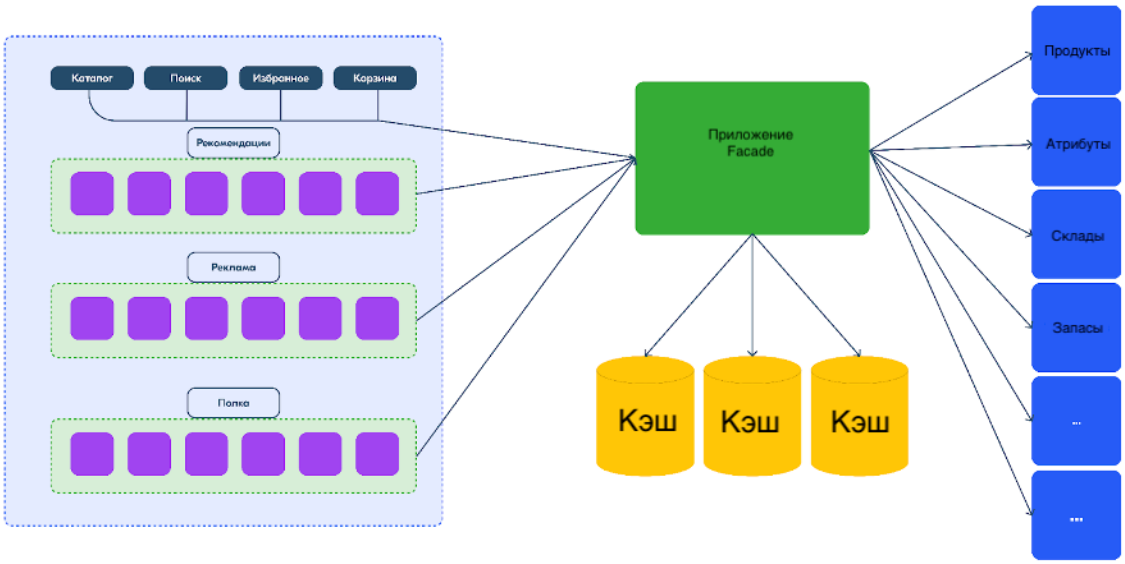

Рис. 1. Архитектура серверного взаимодействия

В следующих разделах мы рассмотрим более детально стратегии кэширования, принципы инвалидации данных и другие аспекты, гарантирующие

бесперебойную работу facade в условиях высокой нагрузки и динамичного онлайн-торгового пространства.

Эволюция кэширования

На заре развития facade мы столкнулись с необходимостью выбора подходящего внешнего хранилища для кэша [5]. По ряду факторов, таких как простота использования и опыт работы с ним в компании, был выбран memcached. Этот выбор обусловлен не только техническими аспектами, но и доступностью опыта работы, что сыграло важную роль в успешной реализации.

Стратегия, которую мы применили на этапе старта, известна как "ленивое кэширование" или Lazy caching [6]. Ее суть заключается в том, что мы обращаемся к кэшу только при наличии запроса от клиента, избегая предварительного кэширования данных. Это подходит под следующую логику:

- Если при запросе в кэш произошла ошибка или ключ не был найден, мы обращаемся к мастер-системе.
- Получив данные, мы отдаем их клиенту и асинхронно записываем их в кэш.

Ниже на рисунке №2 представлен стратегия приложения с “ленивым кешированием”

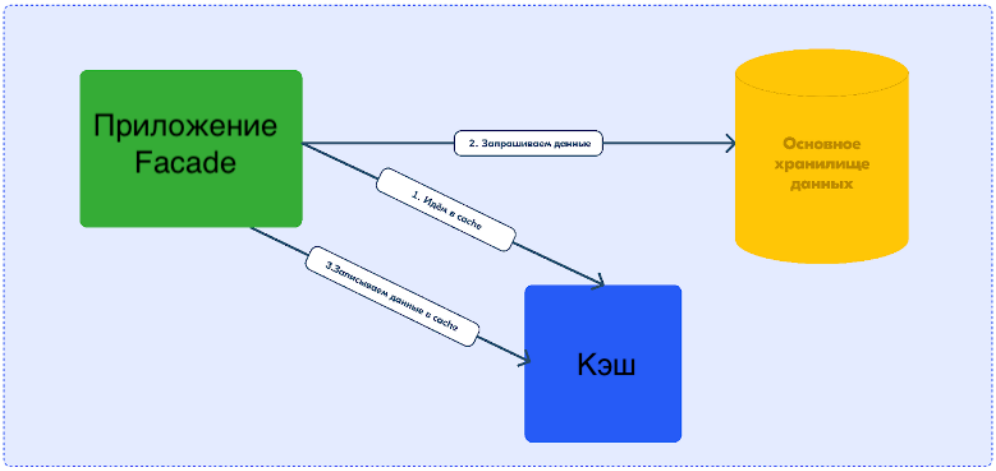

Рис. 2. Стратегия приложения с “ленивым кешированием”

Плюсы “ленивого кеширования”:

- Кэш содержит только запрашиваемые данные: Эффективное использование ресурсов кэша.
- Добавление новых объектов происходит при необходимости: Экономия ресурсов памяти.
- Устойчивость к сбоям кэша: При отсутствии кэша приложение взаимодействует с источником данных напрямую.
- Простая реализация: Легкая интеграция в начальной стадии разработки.

Минусы “ленивого кеширования”:

- Допуск промахов кэша: Возможность несовпадения данных в кэше и мастер-системе.
- Дополнительные задержки при промахе: Каждый промах требует выполнения трех операций.
- Неконсистентность данных: Возможность хранения неактуальных данных в кэше, требующая дополнительных механизмов инвалидации.

Минусы, выявленные в применении ленивого кэширования, побудили нас пересмотреть стратегию удаления и обновления кэша. В начальном этапе мы полагались на TTL (Time-To-Live), устанавливая общий срок жизни на 2 часа для всех объектов кэширования. Этот механизм определял, сколько времени объект оставался в кэше, после чего автоматически удалялся.

Однако, данная модель работала по принципу "eventual consistency" с значительной задержкой. Инвалидация кэша происходила только после истечения установленного TTL. Это означало, что обновленные данные попадали в кэш с существенной задержкой, что могло вызывать неконсистентность данных и ухудшение пользовательского опыта [7].

В следующих разделах мы рассмотрим эволюцию стратегии инвалидации, основанную на событиях и других принципах, для улучшения этой ситуации.

Версионирование и аварийная очистка кеша

Чтобы преодолеть недостатки предыдущего подхода, мы внедрили систему версионирования ключей кэширования. Этот метод стал особенно эффективным, поскольку в нашем случае ключи не зависели друг от друга, что облегчило процесс.

Вся информация в кэше разделена на группы с префиксами, основанными на логическом принципе. Например:

- {item_id}_description_v1
- {item_id}_availability_v1
- {item_id}_attributes_v1

Мы добавили версию к ключу кэширования, например, {item_id}_description_v1. Для инвалидации всех ключей с префиксом "description" достаточно увеличить версию ключа, перейдя, например, от v1 до v2. Мы вынесли управление версиями во внешний конфигурационный сервис, что позволяет изменять версии динамически в рантайме приложения.

При смене версии сервис начинает читать и писать ключи только для новой версии, обновляя все соответствующие ключи при следующем запросе. Таким образом, мы успешно справились с проблемой массового попадания в кэш устаревших данных.

Этот подход не только устраняет недостатки предыдущей стратегии не валидации, но также позволяет внедрять несовместимые изменения в кэшах. Нам удалось более точно и эффективно управлять обновлением данных на витрине, минимизируя задержки и повышая консистентность.

В силу ужесточения требований бизнеса, мы столкнулись с вызовом: необходимость выключения видимости товара на витрине за считанные секунды. Наш текущий сервис не мог справиться с такой задачей, что подвергло сомнению важную функциональность и угрожало срыву запуска. В следующей главе

рассмотрим, как мы решили эту задачу и совершили новый шаг в совершенствовании нашей системы инвалидации кэша.

Инвалидация по событиям через Kafka

Стремясь к максимальной оперативности в обновлении информации на витрине, мы внедрили систему инвалидации кэша на основе событий, получаемых через брокера сообщений Kafka [8]. Сотрудничая с коллегами из систем-источников данных, мы попросили их уведомлять нас обо всех изменениях в товарах.

Получив события от из систем-источников данных, мы теперь можем точно определить, какой кусок данных нужно инвалидировать, и моментально удаляем соответствующий ключ из кэша. Хотя такой подход может привести к дополнительным cache miss-запросам, это не критично для нас. При следующем запросе от клиента мы обращаемся к мастер-системе, получаем актуальные данные и обновляем кэш.

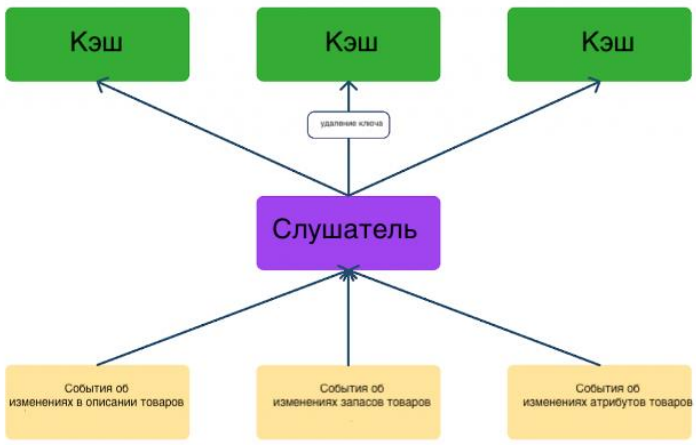

Рис. 3. Инвалидация по событиям через Kafka

Этот метод позволил нам минимизировать задержки при обновлении информации на витрине до нескольких секунд, при условии отсутствия задержек в очереди топика Kafka. К тому же, благодаря этой стратегии, мы смогли увеличить TTL для ключей с 2 часов до 24, повысив при этом Hit ratio — основной показатель эффективности кэширования [9].

$$Hr = {}^{H}/_{T} * 100\ \% \quad (1)$$

Где:

H - количество успешных запросов, удовлетворенных данными из кэша.

T - общее количество запросов к системе.

Таким образом, наша стратегия инвалидации кэша стала трехкомпонентной системой, объединяя TTL, версионирование ключей и принудительную инвалидацию по событиям из Kafka. Даже в случае, если мы не получаем событие об изменении данных, кэш все равно инвалидируется [10] по TTL, гарантируя доставку актуальной информации до витрины.

Формула для оценки эффективности трехкомпонентного кэширования с учетом временных затрат может быть представлена следующим образом:

Этот подход позволил нам стать более стойкими к проблемам неконсистентности данных, а также лучше реагировать на изменения в товарах, обеспечивая более оперативное обновление витрины.

Ниже на рисунке №4 представлен график RPS во время релиза нового функционала.

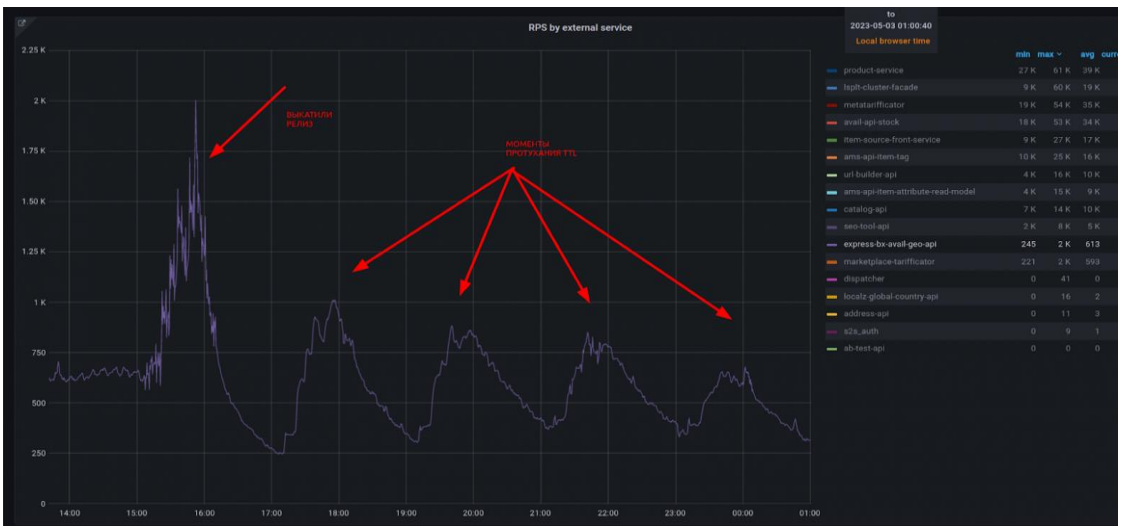

Рис. 4. График RPS

Анализ системы

Необходимо аналитическим способом смоделировать работу системы до и после изменений, рассчитать ее эффективность и надежность.

Анализ системы до изменений

Опишем систему до изменений в виде основных компонент на примере взаимодействия с 1 мастер системой.

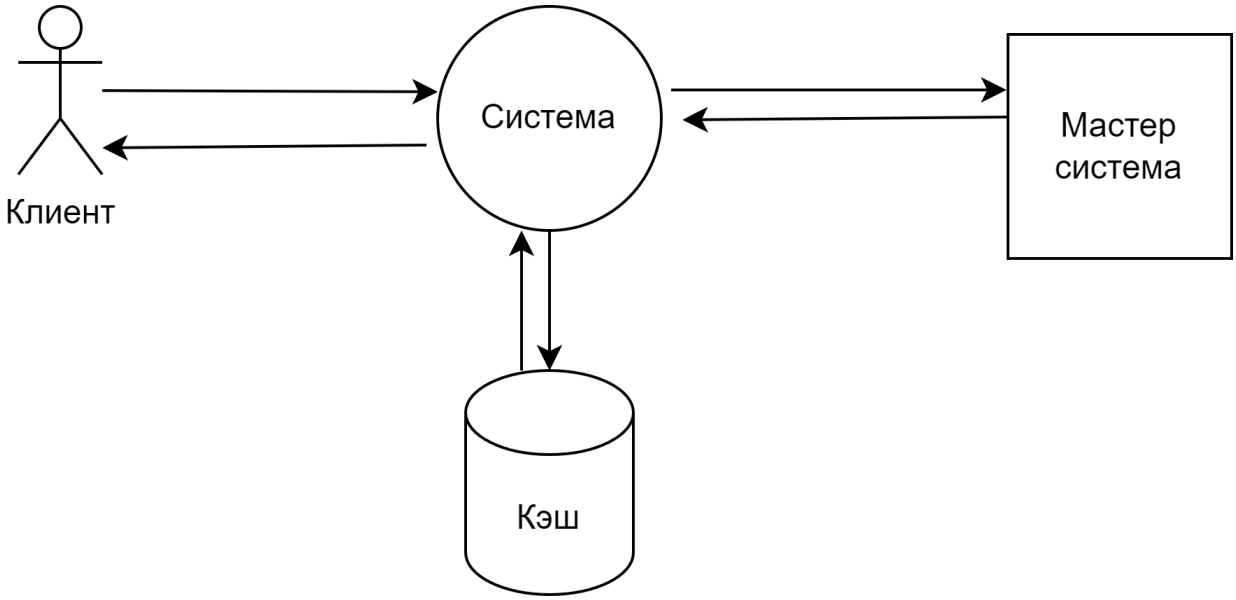

Рис. 5. Система до изменений

Оценим вероятность отказа системы, для этого посчитаем вероятность каждого из компонентов, с которым система взаимодействует, ниже на рисунке №5 представлена вероятность отказа каждого из компонентов.

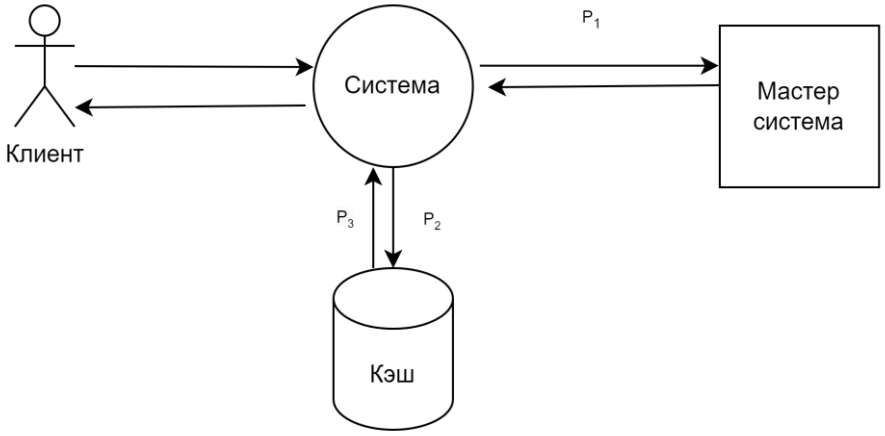

Рис. 6. Система до изменений с вероятностями отказа

Где

$P_1 -$ *вероятность отказа мастер системы*

$P_2 -$ вероятность отказа системы кэширования при записи в нее данных

$P_3 -$ вероятность отказа системы кеширования при получении данных

Распишем каждую из вероятностей отказа компонентов более детально:

$$P_1 = P_{master\ system} \quad (2)$$

$$P_2 = (1 - P_1) * P_{cache\ write} \quad (3)$$

$$P_3 = P_{cache\ get} \quad (4)$$

Построим график последовательности действий системы в контексте вероятности надежности, чтобы оценить общую надежность системы:

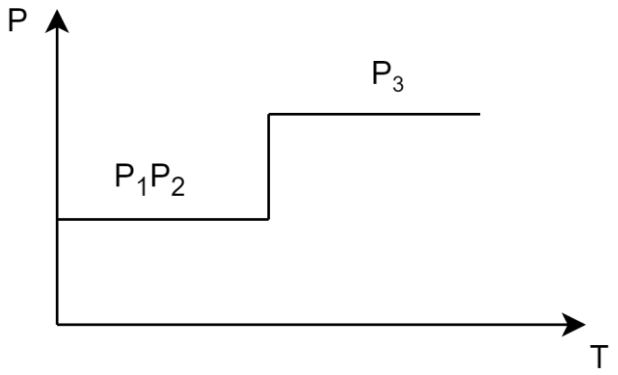

Рис. 7. График последовательности действий сист5емы в контексте вероятности надежности

Из рисунка №7 видно, что вероятности P1 и P2 зависят друг от друга, а вероятность P3 происходит отдельным событием из этого можно сделать вывод, что формула вероятности отказа всей системы будет иметь вид:

$$P = max\{P_1, P_2\} * P_3 \quad (5)$$

одставим в формулу вероятности каждого из компонентов:

$$P = \max\{P_{master\ system}, (1 - P_{master\ system}) * P_{cache\ write}\} * P_{cache\ get} \quad (6)$$

Теперь, когда мы знаем как оценить отказ нашей системы, нам необходимо определить эффективность нашей системы, для этого будем оценивать коэффициент временных затрат, он может быть представлен следующим образом:

$$E_{lazy} = (1 - P_{Cmiss}) * T_{Chit} + P_{Cmiss} * (T_{Mrequest} + T_{Cinvalidation}) \quad (7)$$

Где:

E_{lazy} - коэффициент временных затрат.

P_{Cmiss} - вероятность промаха кэша.

T_{Chit}- время выполнения запроса данных из кэша при попадании.

$T_{Mrequest}$- среднее время на запрос данных из основного источника при промахе кэша.

$T_{Cinvalidation}$- среднее время на инвалидацию кэша.

Эта формула учитывает вероятность попадания в кэш и время выполнения операций как при успешном попадании в кэш, так и при промахе.

Анализ системы после изменений

Опишем систему после изменений в виде основных компонент на примере взаимодействия с 1 мастер системой.

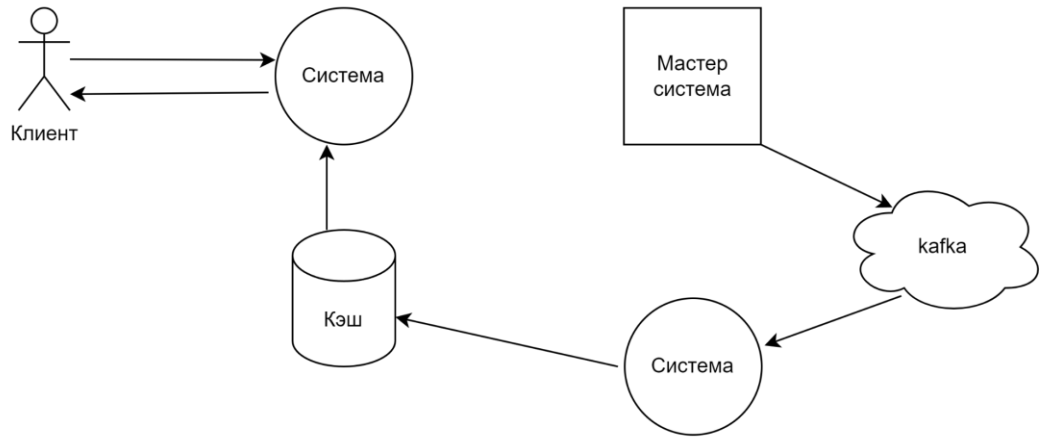

Рис. 8. Система после изменений

Оценим вероятность отказа системы после изменений, для этого посчитаем вероятность каждого из компонентов, с которым система взаимодействует, ниже на рисунке №9 представлена вероятность отказа каждого из компонентов.

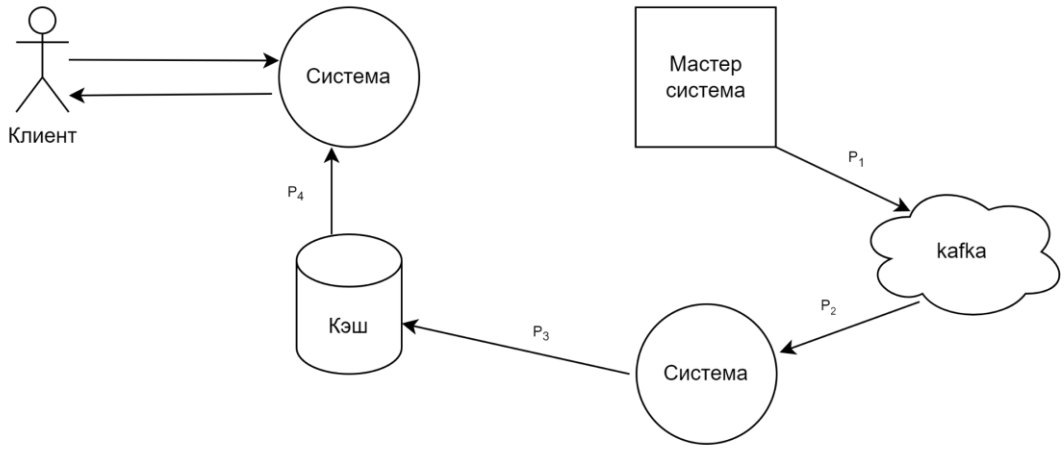

Рис. 9. Система после изменений с вероятностями отказа

Где

$P_1 -$ *вероятность отказа мастер системы*.

$P_2 -$ *вероятность отказа системы* $kafka$.

$P_3 -$ *вероятность отказа системы кэширования при записи в нее данных*.

$P_4 -$ *вероятность отказа системы кеширования при получении данных*.

Распишем каждую из вероятностей отказа компонентов более детально:

$$P_1 = P_{master\ system} \quad (8)$$

$$P_2 = (1 - P_1) * P_{kafka} \quad (9)$$

$$P_3 = (1 - P_1) * \left(1 - P_{kafka}\right) * P_{cache\ write} \quad (10)$$

$$P_4 = P_{cache\ get} \quad (11)$$

Построим график последовательности действий системы в контексте вероятности надежности, чтобы оценить общую надежность системы:

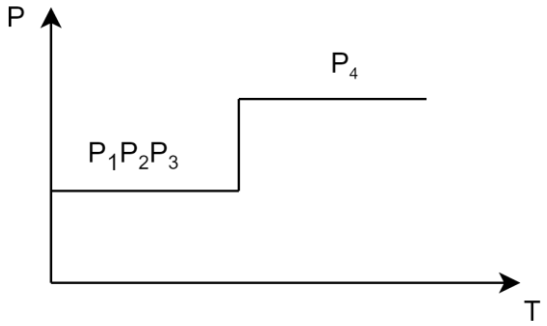

Рис. 10. График последовательности действий системы после изменений в контексте вероятности надежности

Из рисунка №10 видно, что вероятности P1, P2, P3 зависят друг от друга, а вероятность P4 происходит отдельным событием из этого можно сделать вывод, что формула вероятности отказа всей системы будет иметь вид:

$$P = max\{P_1, P_2, P_3\} * P_4 \quad (12)$$

Подставим в формулу вероятности каждого из компонентов:

$$P = max\left\{\begin{matrix} P_{master\ system}, \left(1 - P_{master\ system}\right) * P_{kafka}, \left(1 - P_{master\ system}\right) \\ * \left(1 - P_{kafka}\right) * P_{cache\ write} \end{matrix}\right\} * P_{cache\ get} \quad (13)$$

Теперь, когда мы знаем, как оценить отказ нашей системы после изменений, нам необходимо определить эффективность нашей системы после изменений.

$$E_{trio} = (1 - P_{miss}) * T_{hit} + P_{miss} * T_{Mrequest} \quad (14)$$

Где:

E_{trio} - коэффициент временных затрат.

P_{miss} - вероятность промаха кэша.

$T_{Mrequest}$ - среднее время на запрос данных из основного источника при промахе кэша.

Сравнивая формулу трехкомпонентного кеширования с формулой Ленивого кеширования, то можно заметить, что в новой нет части инвалидации даных в кеше, так как это стало асинхронным процессом, работающим по событиям из Kafka.

Сравнение показателей

Подставим значения вероятностей и временных показателей в зависимости от нагрузки на сервис в формулы для коэффициента временных затрат и вероятности надежности системы и проанализируем результаты. Ниже в таблице № 1 представлены зафиксированные параметры системы.

Таблица 1. Параметры системы

Нагрузка	100 rps	500 rps	1000 rps
T cache get	10	20	40
P miss	0,3	0,5	0,6
T master system	500	900	950
T invalidation	10	50	80
P master system	0,2	0,3	0,4
P cache write	0,09	0,2	0,3
P cache get	0,1	0,3	0,4
P kafka	0,3	0,55	0,6

Ниже представлены результаты вероятности отказа системы на графике и в таблице №2.

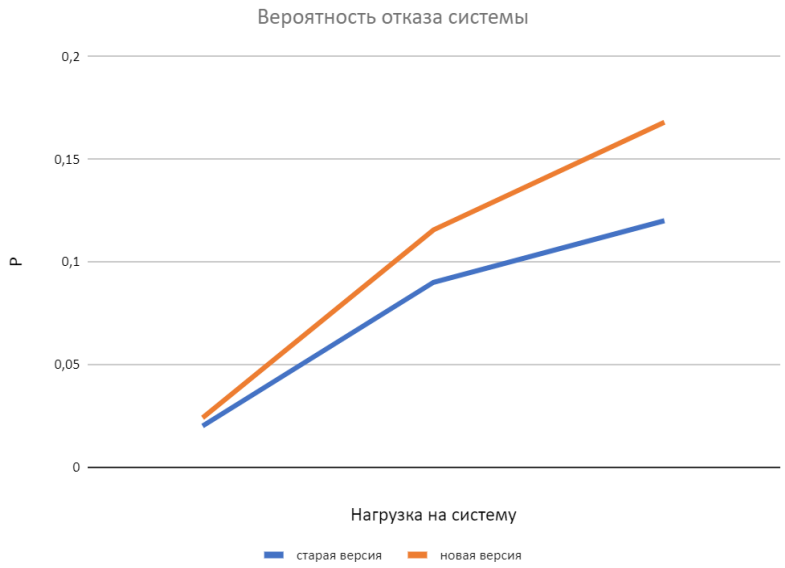

Рис. 11. Вероятность отказа системы

Таблица 2. Вероятность отказа системы

Вероятность отказа системы		
Старая	**Новая**	**%**
0,02	0,024	20
0,09	0,1155	28,333
0,12	0,168	40

Исходя из полученных результатов, можно сделать вывод, что вероятность отказа системы выросла на 20% при малых нагрузках и на 40% при высоких. Рассмотрим, как изменился коэффициент затрат.

Рис. 12. Нагрузка на систему

Таблица 3. Коэффициент временных затрат

Коэффициент временных затрат		
Старый	**Новый**	**%**
160	157	1,875
485	460	5,154
634	586	7,570

На основании анализа полученных результатов становится ясно, что система с интегрированной инвалидацией кэша через Kafka проявляет значительные преимущества в сокращении временных затрат. На фоне низкой нагрузки мы наблюдаем снижение коэффициента временных затрат на 2%, а при высоких нагрузках — на 7,5%. Это свидетельствует о том, что реорганизация системы, включающая в себя механизм инвалидации кэша через Kafka, оказалась эффективной.

Однако, следует отметить, что вместе с улучшением эффективности произошло падение надежности системы. Использование асинхронных событий

из Kafka, несмотря на свою высокую эффективность в обновлении данных, вносит элемент непредсказуемости.

Таким образом, между повышением эффективности и снижением надежности существует компромисс, который следует учитывать при принятии решения о конкретной стратегии инвалидации кэша. В контексте данного исследования, снижение временных затрат при использовании Kafka подчеркивает важность оценки потребностей системы и адаптации стратегий для достижения оптимального баланса между эффективностью и надежностью.

Заключение

Внедрение тройной системы инвалидации кэша, основанной на TTL, версионировании ключей и принудительной инвалидации по событиям из Kafka, привело к значительным улучшениям в работе нашего сервиса. Мы достигли увеличения эффективности и скорости обновлений, минимизации задержек и повышения стабильности даже в условиях экстремальных нагрузок. Этот подход также позволил нам быстрее реагировать на изменения, уменьшить неконсистентность данных и сократить время доставки данных до витрины.

Однако, сделанные жертвы в надежности оказались оправданными, учитывая выигрыш в производительности. Все это подчеркивает важность взвешенного подхода к оптимизации системы, учитывая бизнес-требования и контекст использования, и позволяет обеспечить более эффективную и конкурентоспособную работу в современной среде.

Литература

1. Костиков Ю. А. и др. Применение современных технологий, подходов и шаблонов проектирования при реализации масштабируемой клиент-серверной информационной системы //актуальные научные проблемы прикладных и естественных наук. – 2018. – С. 48-55.
2. Кутузов К. О. Программирование RESTful приложений на языке программирования Golang //Молодость. Интеллект. Инициатива. – 2021. – С. 23-24.

3. Борисов А. Л., Борисов С. Ю. Повышение эффективности использования вычислительных ресурсов корпоративных информационных систем с помощью технологии распределённых вычислений //Вестник ТвГТУ. – 2012. – Т. 180. – №. 20. – С. 3-6.
4. Fitzpatrick B. Distributed caching with memcached //Linux journal. – 2004. – Т. 2004. – №. 124. – С. 5.
5. Грушин Д. А., Лазарев Д. О., Фомин С. А. Кэширование данных в мультиконтейнерных системах //Труды Института системного программирования РАН. – 2019. – Т. 31. – №. 6. – С. 125-144.
6. Шуст И. В., Горлушкина Н. Н. Выбор стратегии кэширования данных в реализации проектов мобильных приложений для работы с клиентами //Экономика. Право. Инновации. – 2023. – №. 1. – С. 63-70.
7. Синчурина М. Г., Шипицына Н. В. Дифференциация понятий «пользовательский интерфейс»(UI) и «пользовательский опыт»(UX) //Коммуникационные технологии: социально-экономические и информационные аспекты. – 2019. – С. 203-205.
8. Wang Z. et al. Kafka and its using in high-throughput and reliable message distribution //2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS). – IEEE, 2015. – С. 117-120.
9. Berger D. S. et al. Maximizing cache hit ratios by variance reduction //Acm sigmetrics Performance Evaluation Review. – 2015. – Т. 43. – №. 2. – С. 57-59.
10. Гурьянова А. А. Разработка подсистемы кэширования данных в облачной системе //международная молодежная научная конференция" xxii туполевские чтения (школа молодых ученых)". – 2015. – С. 405-409.

Reference

1. Kostikov Yu. A. et al. Application of modern technologies, approaches and design patterns in the implementation of a scalable client-server information system //actual scientific problems of applied and natural sciences. - 2018. – pp. 48-55.

2. Kutuzov K. O. Programming of RESTful applications in the Golang programming language //Youth. Intelligence. Initiative. - 2021. – pp. 23-24.
3. Borisov A. L., Borisov S. Y. Improving the efficiency of using computing resources of corporate information systems using distributed computing technology //Bulletin of TvSTU. – 2012. – Vol. 180. – No. 20. – pp. 3-6.
4. Fitzpatrick B. Distributed caching with memcached //Linux journal. – 2004. – Vol. 2004. – No. 124. – p. 5.
5. Grushin D. A., Lazarev D. O., Fomin S. A. Data caching in multi-container systems //Proceedings of the Institute of System Programming of the Russian Academy of Sciences. – 2019. – vol. 31. – No. 6. – pp. 125-144.
6. Shust I. V., Gorlushkina N. N. Choosing a data caching strategy in the implementation of mobile application projects for working with clients //Economy. Right. Innovation. - 2023. – No. 1. – pp. 63-70.
7. Sinchurina M. G., Shipitsyna N. V. Differentiation of the concepts of "user interface"(UI) and "user experience"(UX) //Communication technologies: socio-economic and information aspects. – 2019. – pp. 203-205.
8. Wang Z. et al. Kafka and its using in high-throughput and reliable message distribution //2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS). – IEEE, 2015. – pp. 117-120.
9. Berger D. S. et al. Maximizing cache hit ratios by variation reduction //Acm sigmetrics Performance Evaluation Review. – 2015. – vol. 43. – No. 2. – pp. 57-59.
10. Guryanova A. A. Development of a data caching subsystem in a cloud system //international youth scientific conference"xxii tupolev readings (school of young scientists)". – 2015. – PP. 405-409.

Для цитирования: Горячкин Б.С., Зудин А.М. Анализ серверных вычислений// Научный сетевой журнал «Столыпинский вестник» №5/2024