МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт Математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Управление корпоративными информационными процессами
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Исследование возможностей использования технологии Conflict-free Replicated Data Types (CRDT) в разработке горизонтально масштабированных веб-приложений»

Обучающийся Н.А. Кальянов
(И.О. Фамилия) (личная подпись)

Научный руководитель к.т.н., доцент, А.Б. Кузьмичёв
(ученая степень, звание, И.О. Фамилия)

Тольятти 2022

Оглавление

Введение

Актуальности темы исследования: С каждым годом Интернет все больше проникает в различные сферы экономики [8]. Для существования на современном рынке практически каждой организации необходимо создавать и поддерживать веб-приложения. Поскольку разработка и поддержка веб-приложений представляет собой непростую задачу, в настоящий момент для ее упрощения было выработано несколько подходов и конкретных программных продуктов (фреймворки, конструкторы приложений и т. д.). Тем не менее, появляются новые требования пользователей к отказоустойчивости и синхронизации данных — возникает необходимость разрабатывать веб-приложения с поддержкой распределенного хранения данных с оперативной синхронизацией. К реализации подобной синхронизации можно подойти по-разному, с использованием различных алгоритмов балансировки нагрузки и репликации (например, использование балансировщиков нагрузки, использование распределенных СУБД типа master-master или синхронной репликацией). Однако, построение корректных и производительных распределенных систем хранения данных является одной из сложнейших задач прикладной информатики. В связи с этим наиболее актуальной является задача использования новейших теоретических достижений в разработке и поддержке веб-приложений на практике.

Научная проблема: констрадикция научной проблемы проявляется в том, что современные рыночные условия требуют отказоустойчивых веб-приложений с динамической синхронизацией состояния между сессиями, что требует распределенного хранения данных. При этом проектирование и поддержка распределенной системы хранения данных является сложной задачей, требующей значительных ресурсов и поэтому часто является

недоступной для малых организаций. В то же время, существуют новые подходы к синхронизации данных, которые могут обеспечивать математически доказанную корректность синхронизации а также потенциально более высокую производительность и простоту разработки и поддержки. Одним из перспективных семейств алгоритмов являются алгоритмы CRDT. Однако, несмотря на их перспективность в настоящее время алгоритмы CRDT слабо внедрены в практическую разработку типичных веб-приложений.

Финитизация данной научной проблемы заключается в радикальном упрощении разработки и поддержки широкого спектра типичных веб-приложений, используемых в различных сферах экономики; удешевлении процессов поддержки веб-приложения и повышении производительности веб-приложений.

Цель исследования: разработать подходы к практическому внедрению алгоритмов CRDT в современные веб-приложения, оценить корректность и производительность полученной реализации по сравнению с альтернативными реализациями и существующими решениями в области хранения данных.

Область исследования: архитектура веб-приложений, алгоритмы синхронизации данных, распределенные системы хранения данных.

Гипотеза: процесс разработки и поддержки современных веб-приложений будет значительно упрощён и удешевлён если будут решены следующие задачи:

- Проанализированы актуальные подходы и программные решения для распределенного хранения и синхронизации данных.

- Изучены текущее использование алгоритмов CRDT в различных видах веб-приложений.

- Разработаны стратегии внедрения CRDT в распространённые типы веб-приложений.

- Разработана Proof-of-Concept реализация CRDT

- оценена производительность, скорость внедрения и корректность полученной реализации по сравнению с альтернативными решениями.

Задачи исследования:

Проанализировать текущее состояние предметной области.

Разработать подходы к внедрению алгоритмов CRDT в различные типы веб-приложений (включая типичные веб-приложения для малого и среднего бизнеса).

Проанализировать свойства практической реализации алгоритмов CRDT по сравнению с альтернативными решениями.

Практическая значимость: Упрощение и удешевление разработки отказоустойчивых распределенных веб-приложений для широкого круга организаций (включая малый и средний бизнес) в соответствии с требованиями пользователей и контрагентов. Упрощение и удешевление эксплуатации полученных веб-приложений с распределенным хранением данных.

Глава 1 Анализ проблем распределенной синхронизации данных

1.1 Анализ проблем распределенной синхронизации данных в веб-приложениях

По результатам анализа литературы было обнаружено три актуальных проблемы современных веб-приложений. Так, в статье «Интеграция распределенных корпоративных приложений: исследование» сообщается о трендах корпоративных приложений: рыночные условия вынуждают организации разрабатывать и поддерживать распределенные (в том числе географически) приложения [20]. Эти же тенденции замечают и компании-провайдеры инфрастуктуры — о том, что современные корпоративные приложения все активнее используют распределенное хранение данных сообщает компания «F5 Networks Inc.» [2]. В качестве причин указывается давление рынка, а именно требования по скорости ответа, отказоусточивости и безопасности данных. Обнаруженные проблемы представлены в таблице 1.

Таблица 1 – Актуальные проблемы синхронизации данных современных веб-приложений

№ п/п	Проблема	Причины проблемы	Сфера возникновения
1	Невозможно реализовать эффективную географически распределенную синхронизацию данных	- синхронизация данных по географически распределенным дата-центрам может вызвать проблемы производительности из-за сетевых задержек	Веб-приложения крупных корпораций а также веб-приложений корпораций, бизнес которых требователен к обеспечению отказоусточивости и производительности

Продолжение таблицы 1

№ п/п	Проблема	Причины проблемы	Сфера возникновения
	Сложно обеспечить синхронизацию данных в горизонтально масштабированных веб-приложениях	- синхронизация данных между несколькими отказоустойчивыми экземплярами веб-приложения может вызывать проблемы производительности	Веб-приложения крупных корпораций а также веб-приложений корпораций, бизнес которых требователен к обеспечению отказоусточивости
	Сложно избежать потери данных при синхронизации клиентских сессий на мобильных устройствах	- синхронизация данных между несколькими сессиями, каждая из которых может отключится в любой момент может привести к потере данных	Кросс-платформенные приложения, включая мобильные приложения

Первая проблема возникает в современных веб-приложениях больших корпораций, где приходится иметь несколько распределенных дата-центров в силу размера и требований к производительности. Кроме того, один единственный экземпляр веб-приложения может не отвечать требованиям по скорости отклика, так как находится в конкретной географической локации, не обязательно оптимальной для каждого потребителя сервисов. Таким образом, давление технических ограничений приводит к необходимости поддерживать распределенную инфраструктуру и соответственно обеспечивать синхронизацию данных между узлами.

Вторая проблема проявляется в любых достаточно популярных веб-приложениях. Из-за роста проникновения Интернета во все сферы жизни [36], роста числа подключённых устройств, развития Интернета Вещей (IoT, Internet-of-Things) один единственный экземпляр веб-приложения попросту физически не способен обслуживать столь большое число запросов. Как известно, существует два основных вида масштабирования приложений — вертикальное масштабирование и горизонтальное масштабирование.

Вертикальное масштабирование заключается в наращивании мощности сервера приложений и/или сервера базы данных, так, чтобы более производительные аппаратные ресурсы помогали выдерживать увеличенные нагрузки. У этого вида масштабирования есть как технические ограничения (например, невозможно бесконечно повышать частоту процессора), так и экономические ограничения. Давление экономических ограничений более сильное так как стоимость аппаратных ресурсов в зависимости от их производительности растёт не линейно.

Эта же проблема проявляется в веб-приложениях корпораций (не обязательно крупных), бизнес-специфика которых требует обеспечения отказоусточивости (например, облачные услуги). Стабильные и высокодоступные сервисы могут с одной стороны являться конкурентным преимуществом на рынке, с другой стороны если большинство компаний-конкурентов обладают отказоусточивостью, то обеспечение как минимум аналогичного уровня отказоустойчивости (например в терминах SLA, Service Level Agreement) становится обязательным условием для выживания компании на рынке. Как известно, основными механизмами обеспечения отказоустойчивости являются резервирование и создание избыточности — приходится поддерживать физические изолированные дата-центры. То есть под давление бизнес-требований возникает проблема обеспечения согласованности данных в разных дата-центрах и их оперативной синхронизации.

Третья проблема также связана с синхронизацией данных, однако в этом случае речь идёт скорее о клиентской части веб-приложения (например, мобильные приложения). В настоящее время все чаще требуется синхронизировать данные от сессий пользователя с разных устройств, так как средний пользователь использует все больше и больше подключенных к

Интернету устройств (особенно значительный рост наблюдается у сегмента смартфонов в развивающихся странах [31], а также устройств Интернета вещей (Internet of Things, IoT) [34]). При этом часть сессий может терять соединение с серверами компании (например, при использовании мобильного интернета в движении). Возникает потребность корректно синхронизировать состояние на нескольких устройствах даже с учётом возможных обрывов и последующих восстановлений соединения, чтобы избежать потери пользовательских данных.

Таким образом, было выявлено три проблемы, возникающих при разработке современных веб-приложений. Первая проблема заключается в невозможности реализовать эффективную синхронизацию данных между географически распределенными дата-центрами крупных корпораций. Вторая проблема — сложно реализовать надёжную и производительную синхронизацию данных для горизонтально масштабируемых веб-приложений. Третья проблема — трудно избежать потери пользовательских данных при параллельном использовании нескольких мобильных клиентов и нестабильном интернет-соединении. По критериям практической распространённости правильнее рассматривать проблему 2 — эффективное обеспечение синхронизации данных горизонтально масштабированных веб-приложений.

1.2 Анализ существующих подходов к решению проблемы эффективной синхронизации данных горизонтально масштабированных веб-приложений

Давление с двух сторон, как было показано в 1.1 (давление технических ограничений и давление бизнеса) приводит современные организации к необходимости обеспечивать распределенную синхронизацию данных между горизонтально масштабированными экземплярами веб-

приложения. Ниже представлены возможные подходы к решению проблемы обеспечения эффективной синхронизации данных между экземплярами веб-приложений [15].

Распределенные транзакции. В данном подходе для синхронизации используются распределенные транзакции с двухшаговым протоколом по стандарту X/Open XA [39]. В стандарте определено 3 роли — приложение, менеджер транзакций и менеджер ресурсов. Приложение определяет границу транзакции и делегирует координацию распределенной транзакции менеджеру транзакции. Менеджер транзакции координирует несколько менеджеров ресурсов (например, несколько серверов баз данных) с использованием двухшагового алгоритма, представленного на рисунке 1.

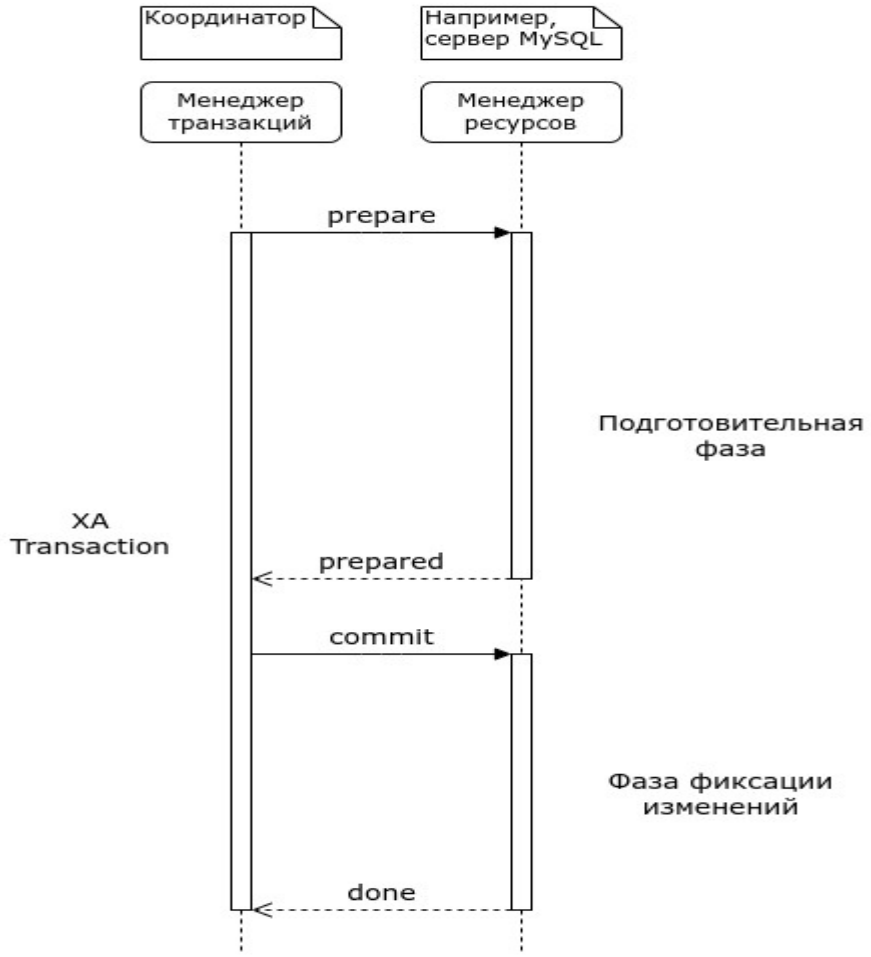

Рисунок 1 – Двухшаговый процесс распределенных транзакций по стандарту X/Open XA

В частности, менеджер транзакций следит за тем, чтобы все серверы баз данных успешно завершили подготовку к совершению транзакции. Если подготовительный этап прошел успешно на всех серверах менеджер транзакций может зафиксировать (commit) транзакцию на всех серверах или откатить (rollback) транзакцию при возникновении ошибок. Общая пошаговая схема распределенной синхронизации данных с использованием данного подхода представлена на рисунке 2.

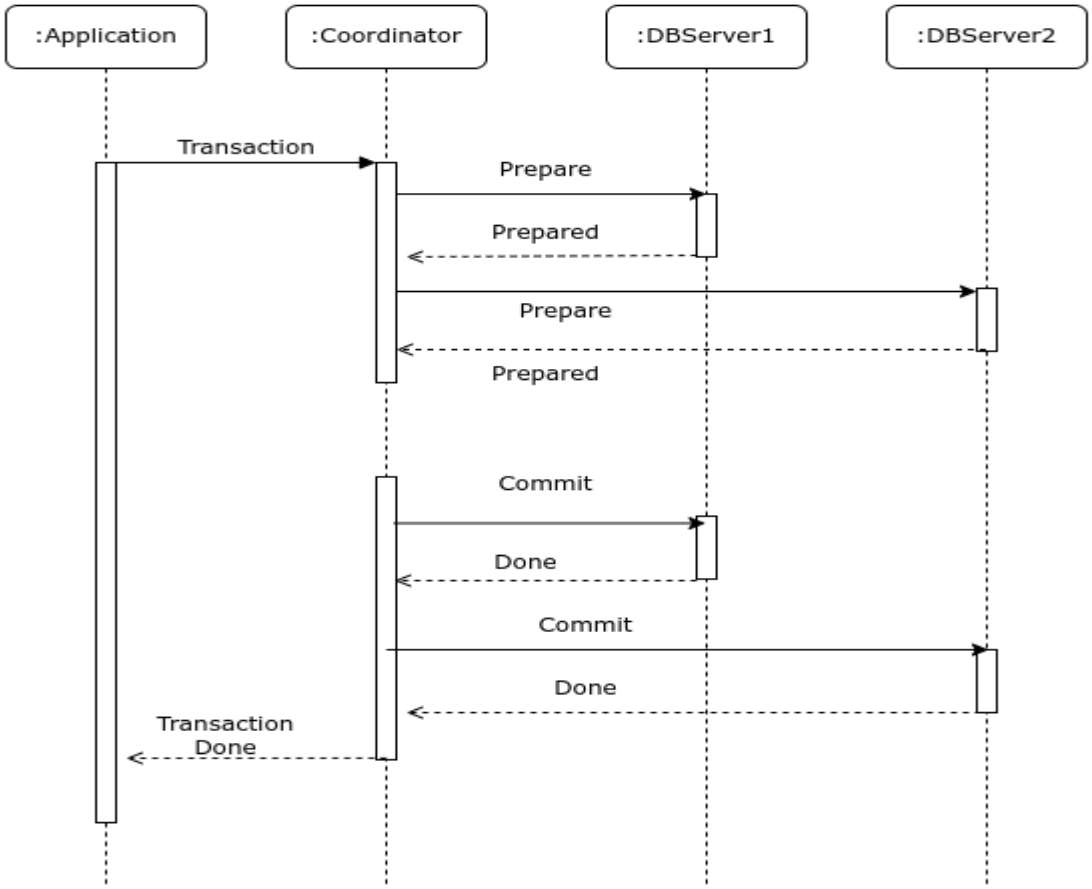

Рисунок 2 – Распределенная синхронизация данных с использованием распределенных транзакций

На первом шаге приложение запрашивает у координатора совершение транзакции и передаёт содержимое транзакции. Координатор запрашивает подготовку ресурсов у каждого из двух серверов баз данных и дожидается подтверждения.

На втором шаге координатор принимает решение о фиксации изменений, отправляет запросы на фиксацию изменений на каждый из двух участвующих серверов баз данных и дожидается подтверждения. После получения обоих подтверждений координатор отвечает приложению об успешном завершении распределенной транзакции. В этот момент данные полностью синхронизированы на обоих серверах.

Поддержка стандарта присутствует во многих популярных СУБД, что облегчает поддержку. Также этот подход обеспечивает строгую согласованность данных. В то же время при использовании данного подхода могут возникнуть проблемы с производительностью — каждое изменение данных будет приводить к нескольким распределенным блокировкам, которые являются относительно «дорогими» операциями. В данном примере оба сервера баз данных должны поддерживать блокировки на всем протяжении времени между подготовительной фазой и фазой фиксации изменений чтобы обеспечивать корректность данных в условиях возможных других одновременных запросов на изменения. Кроме того, это решение может страдать от сетевых задержек.

Специализированные кластерные БД. В данном подходе используются специализированные кластерные решения, способные обеспечить строгую согласованность данных и приемлемую производительность. Примером подобного решения является Galera MySQL — кластерное решение на основе протокола WSREP поверх популярной открытой СУБД MySQL/MariaDB [16].

В то же время подобные решения относительно сложно администрировать — каждое решение обладает своими особенностями. Таким образом требуются узкие специалисты что увеличивает стоимость поддержки. Кроме того, могут даже потребоваться доработки на стороне веб-приложения — например для достижения производительности и возможности записи через все узлы кластера Galera использует «оптимистичные блокировки» что отличается от традиционных подходов реляционных СУБД. Программный код веб-приложения должен быть реализован с учётом этой особенности. Общая схема репликации данных в кластере Galera представлена на рисунке 3.

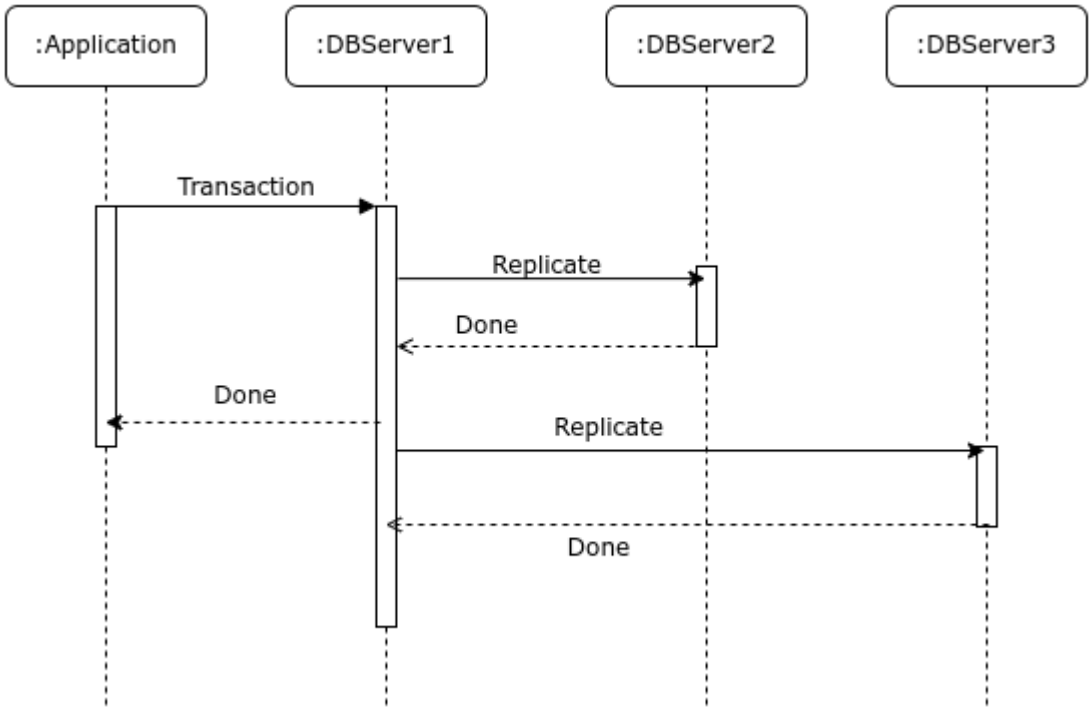

Рисунок 3 – Общая схема синхронизации данных в кластере Galera

Особенность синхронизации данных в кластере Galera — репликация данных псевдосинхронная в том смысле, что не происходит ожидания подтверждения записи данных на других узлах кластера. По этой причине позднее может возникнуть конфликт при репликации данных на другой узел и приложение получит ошибку и будет вынуждено повторить транзакцию.

Стоит отметить, что работа многих кластерных БД основана на динамическом выборе ведущего узла на основе конкретного алгоритма выбора лидера (leader-election algortithm). Для обеспечения устойчивости системы при потере соединения с ведущим узлом другой узел может относительно оперативно взять на себя функции лидера. Ведущий узел отвечает за распределение данных по другим узлам а также за определение итогового результат транзакции с помощью алгоритма консенсуса (например, транзакция может считаться успешной если данные были

успешно записаны более чем на половине узлов кластера). Например, таким образом работает популярное распределенное хранилище «ключ-значение» etcd [13]. Подобное решение позволяет снизить количество распределенных блокировок, так как фактически запись осуществляется через один выделенный узел кластера ценой чуть меньшей отказоустойчивости. Чаще подобные системы являются нереляционными. Общая схема запроса на запись в кластере etcd представлена на рисунке 4.

Приложение осуществляет запись через текущего лидера кластера. Лидер берет на себя роль координатора — реплицирует запись по другим узлам кластера и дожидается подтверждения. После получения необходимого числа подтверждений лидер отвечает приложению об успехе записи.

Таким образом гарантируется синхронизированность данных на всех узлах кластера.

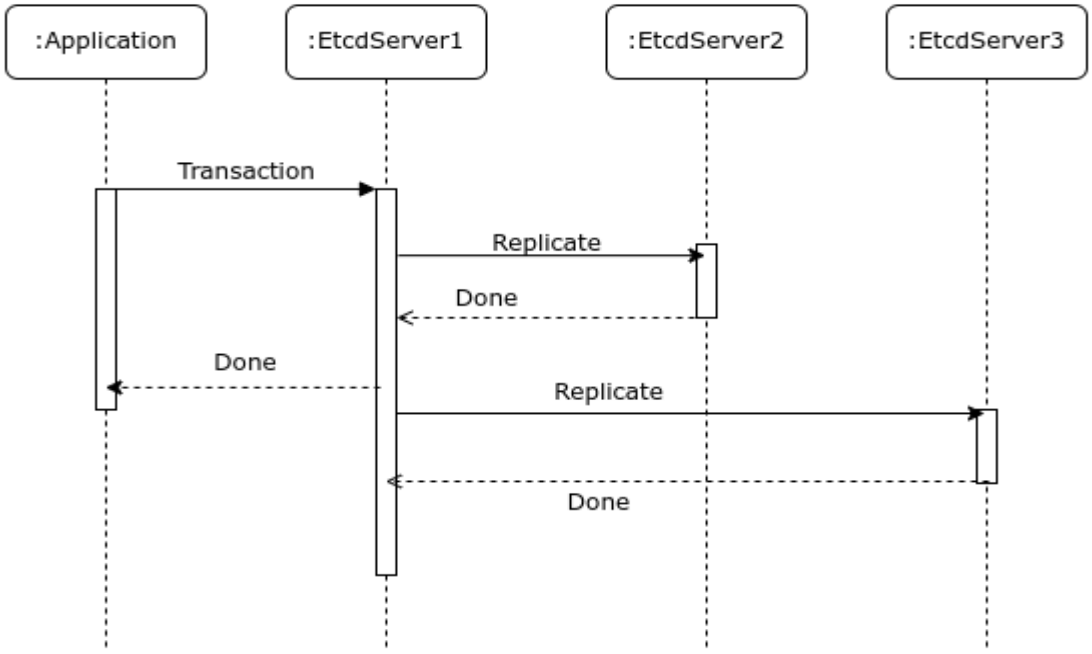

Рисунок 4 – Общая схема проведения записи в кластере etcd

Если часть узлов кластера были недоступны на момент записи, то эти узлы могут получить обновления от лидера как только восстановят соединение. Следует отметить, что приложению не обязательно знать текущего лидера кластера — можно осуществлять запросы через любой узел кластера, а этот узел автоматически перенаправит запрос текущему лидеру кластера.

Асинхронная репликация. В данном подходе используется функционал асинхронной репликации, встроенный во многие популярные СУБД (например, MySQL, PostgreSQL). За счёт этого достигается низкая стоимость поддержки. Асинхронная репликация не использует блокировки, что обеспечивает высокую производительность. Однако, в рамках этого решения не обеспечивается строгая согласованность данных — асинхронная реплика может иметь неактуальные данные и отставание может расти при высокой нагрузке на основной узел. Общая схема процесса синхронизации данных в рамках данного подхода представлена на рисунке 5.

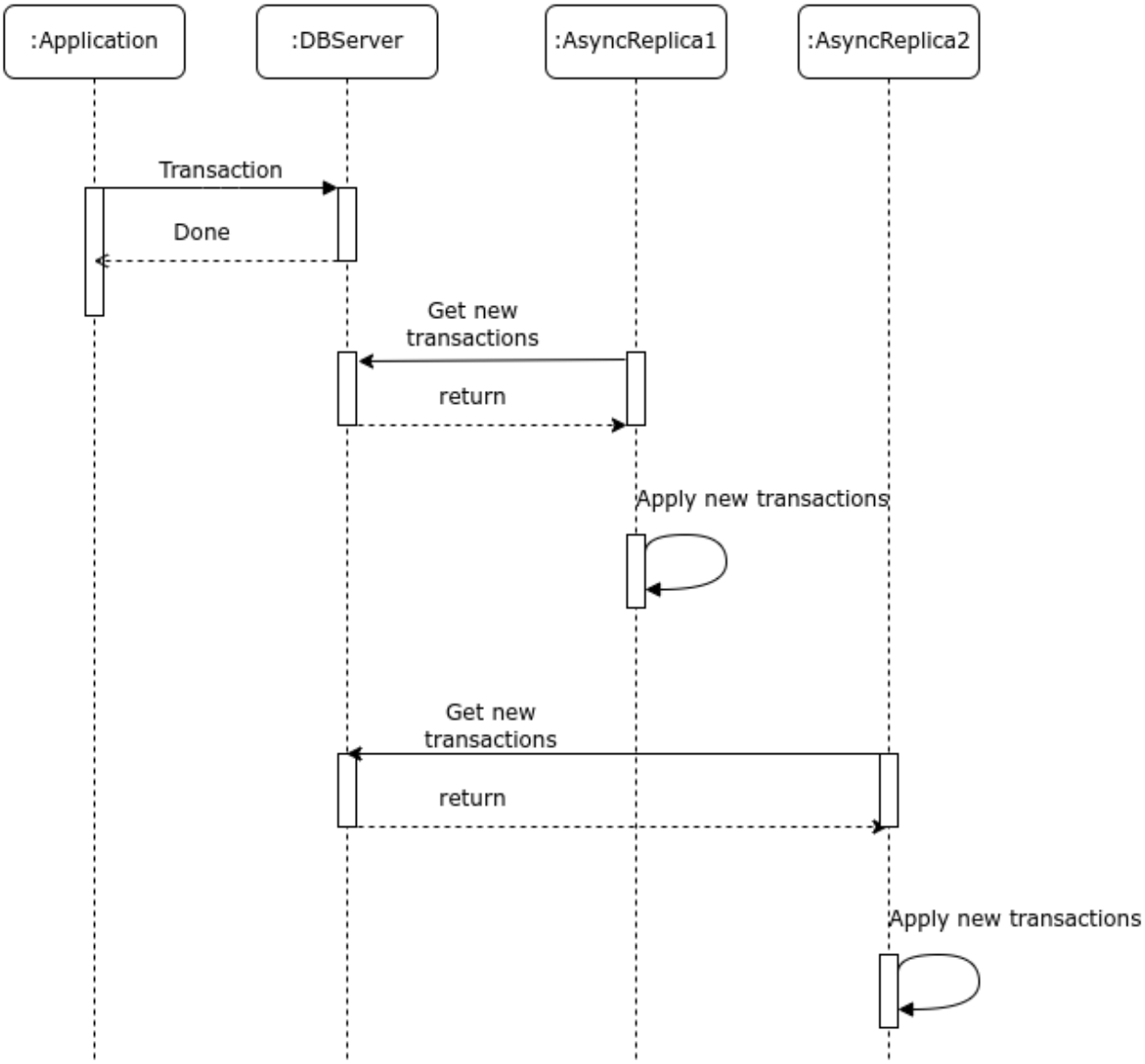

Рисунок 5 – Асинхронная репликация данных

На первом шаге приложение запрашивает совершение транзакции на основном сервере баз дынных, асинхронные реплики напрямую не участвуют в процессе. Основной сервер баз данных применяет транзакцию и отвечает приложению.

На втором шаге асинхронные реплики получают с основного сервера применённые им транзакции и затем повторяют эти транзакции над своими копиями базы данных. Обычно для этого используется журнал событий в

бинарном формате — специальный журнал применённых транзакций в бинарном формате который ведёт основной сервер и читают асинхронные реплики. Лишь после завершения второго шага можно считать данные синхронизированными на всех узлах. При этом при возникновении ошибок синхронизации приложение уже не сможет обнаружить это напрямую. Кроме того, время за которое асинхронные реплики применят свежие транзакции может быть не постоянным — при большой нагрузке на реплики или сетевых проблемах асинхронные реплики могут значительно отстать от основного сервера. Подобная ситуация может требовать ручного вмешательства администраторов инфраструктуры.

Данный подход часто используется для относительно нетребовательных к актуальности данных задач. Например некритичный для бизнеса отчёт может строится по данным на асинхронной реплике, что одновременно позволяет разгрузить основной сервер базы данных.

1.3 Выбор подхода к решению проблемы обеспечения синхронизации данных между горизонтально распределенными экземплярами веб-приложения

Был разработан алгоритм оценки каждого подхода с использованием трёх критериев. Используемые критерии:

- Согласованность данных между экземплярами горизонтально масштабированного веб-приложения;
- Низкая стоимость поддержки выбранного решения синхронизации;
- Производительность и масштабируемость, способность выдерживать высокие нагрузки.

Рассмотрим их подобнее.

1. Согласованность данных. Данный критерий отражает предоставляемые конкретным подходом гарантии согласованности данных.

Чем больше предоставляемые гарантии согласованности, чем проще использовать подход в произвольном веб-приложении. В то же время, нестрогие гарантии согласованности могут требовать особого подхода к разработке веб-приложения или даже быть неприменимыми в некоторых приложениях. Шкала оценки данного критерия была установлена в диапазоне 1-10. Варианты решений проблемы, которые обеспечивают строгую согласованность данных получают максимальный балл по данному критерию. Самый низкий балл по данному критерию (4) получает асинхронная репликация, которая не предоставляет никаких гарантий согласованности данных, но при нормальной работе может обеспечивать достаточно актуальные данные. Использование услуг облачного провайдера может обеспечивать различный уровень в зависимости от конкретного решения, и поэтому получает несколько сниженную оценку (8). Алгоритмы CRDT гарантируют согласованность данных с течением времени, но не строгую согласованность данных в любой момент времени и поэтому получают 6 баллов.

2. Низкая стоимость поддержки. Данный критерий оценивает легкость поддержки конкретного решения. На стоимость поддержки может влиять непосредственно стоимость решения (лицензии, облачные услуги), а также необходимое количество человеко-часов для поддержки решения. Также влияет требуемая квалификации сотрудников для эффективной поддержки решения. Шкала данного критерия, как напрямую влияющая на экономическую эффективность компании установлена в диапазоне 1-15. Максимальный балл по данному критерию получает использование распределенных транзакций за счет широкой поддержки стандарта во многих популярных реляционных СУБД. Близкий к максимальному балл (14) также получает использование алгоритмов CRDT за счёт простоты реализации и

гарантированного отсутствия конфликтов. Несколько сниженный балл (12) получает асинхронная репликация, так как на практике могут возникать отставания реплики, требующие ручного вмешательства администраторов. Самые низкие баллы (3 и 5 соответственно) получает использование специализированных кластерных БД и использование облачных услуг. Первое из перечисленных решений требует узких специалистов и иногда даже доработок приложения, а второе решение может быть экономически неэффективным для многих компаний и также требовать узких специалистов.

3. Производительность. Данный критерий оценивает то, насколько хорошо система выдерживает нагрузку. В том числе учитывается производительность в условиях сетевых проблем, которые могут возникать при реальной эксплуатации. Также учитывается возможность масштабирования системы для улучшения производительности. Шкала данного критерия, как напрямую влияющая на экономическую эффективность компании и пользовательский опыт установлена в диапазоне 1-15. Максимальные баллы по данному критерию получает использование асинхронной репликации, использование услуг облачного провайдера и использование алгоритмов CRDT за счет хорошей масштабируемости перечисленных решений. Использование специализированных кластерных БД получает несколько сниженный балл (12) так может показывать различную производительность в зависимости от конкретного сценария использования. Низкий балл (2) получает использование распределенных транзакций из-за требовательности к скорости сетевого соединения и большого числа распределенных блокировок.

Результаты оценивания вариантов решения проблемы синхронизации данных горизонтально масштабированных веб-приложений с использованием перечисленных критериев приведены в таблице 2.

Таблица 2 – Возможные варианты решения проблемы обеспечения синхронизации данных между экземплярами веб-приложения

Название подхода	Согласованность данных (1-10)	Низкая стоимость поддержки (1-15)	Производительность (1-15)	Итого
Распределенные транзакции [26]	10	15	2	27
Специализированные кластерные БД [16][13]	10	3	12	25
Асинхронная репликация [25] [32]	4	12	15	31
Использование алгоритмов CRDT	6	14	15	35

По сумме оценок видно, что алгоритмы CRDT являются наиболее перспективным подходом к решению проблемы, способным преодолеть некоторые недостатки альтернативных подходов.

Таким образом, были рассмотрены современные подходы к распределенной синхронизации данных: использование распределенных транзакций, специализированные кластерные БД, асинхронная репликация и использование алгоритмов CRDT. Каждый подход обладает как преимуществами, так и недостатками. Иногда недостатки подхода столь значительны, что подход применим лишь для решения ограниченного числа задач. По результатам проведённой оценки по критериям Согласованности данных, Низкой стоимости поддержки и Производительности использование

алгоритмов CRDT является наиболее предпочтительным направлением решения проблемы, далее будет рассматриваться именно этот подход.

1.4 Подход алгоритмов CRDT к решению проблемы синхронизации данных горизонтально масштабированных веб-приложений

Альтернативным подходом является Conflict-free Replicated Data Types (CRDT). CRDT - это относительно новое семейство типов данных (первые исследования по CRDT появились в 2006 году [19]). Они обладают некоторыми примечательными свойствами, благодаря которым становится возможным решить многие из проблем, возникающих при построении распределенных систем. Алгоритмы с использованием CRDT основаны на математических свойствах типов и часто формально верифицируемы, что позволяет исключить возникновение трудноуловимых багов, которые могут возникнуть как в пределах собственной инфраструктуры, так и в инфраструктуре облачных провайдеров. Кроме того, из самого названия (conflict-free), следует, что данное семейство алгоритмов позволяет реплицировать данные “автоматически”, без необходимости разрешения конфликтов, консенсуса, распределенных блокировок и ожидания что позволяет увеличить производительность и значительно упростить разработку и поддержку распределенной системы. Все это становится возможным благодаря свойствам CRDT. Таким образом, алгоритмы CRDT потенциально могут обеспечить «Да» в каждой колонке. Однако, стоит отметить, что CRDT применимы не для всех задач и применимость CRDT должна определятся для конкретной предметной области и конкретного веб-приложения индивидуально. Общая схема репликации данных с использованием алгоритмов CRDT представлена на рисунке 6.

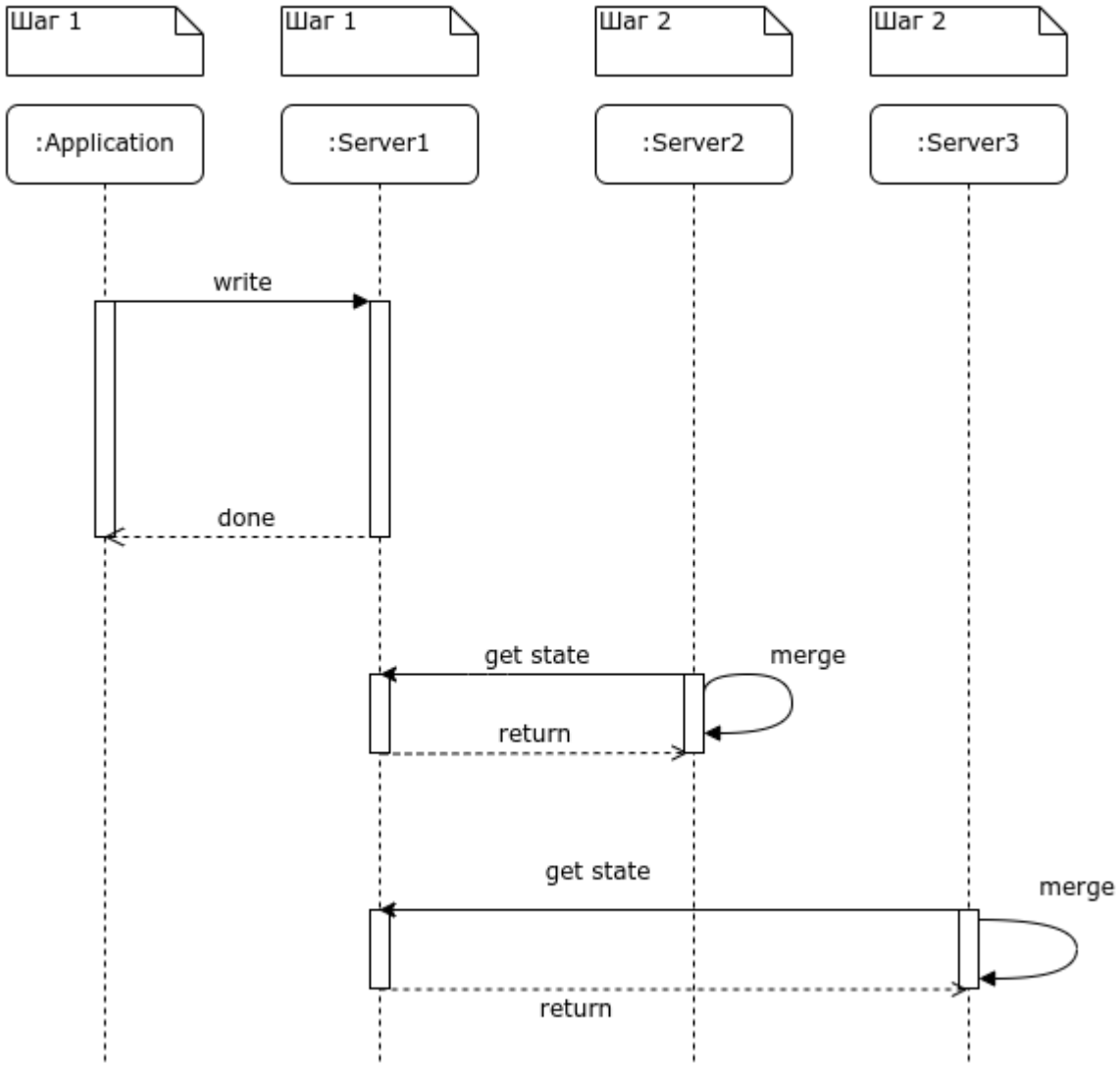

Рисунок 6 – Распределенная синхронизация данных с использованием алгоритмов CRDT

На первом шаге приложение осуществляет запись через любой из узлов кластера, остальные узлы не участвуют в записи.

На втором шаге узлы кластера обмениваются между собой состоянием и таким образом приходят к общему единому синхронизированному состоянию. При этом в отличие от обычной асинхронной репликации, благодаря свойствам алгоритмов CRDT гарантированно не могут возникнуть ошибки репликации при обмене состоянием. Использование алгоритмов CRDT позволяет решить и вторую проблему асинхронной репликации —

возможность отставания реплики. За счёт удобный свойств алгоритмов CRDT обмен данными между узлами не обязательно должен осуществляться с каждого узла. Синхронизация может осуществляться и внешним сервисом, в таком случае нагрузка на узлы кластера не приведёт к отставанию репликации. Кроме того, возможно гибкое управление синхронизацией данных через внедрение синхронизации на сами узлы. Например, каждый узел может при получении запроса на изменение распределять изменения по N соседним узлам, где N может выбираться динамически исходя из требований к согласованности данных конкретного веб-приложения. При такой реализации свойства согласованности данных приближаются к специализированным кластерным решениям типа Master-Master, но в то же время за счёт свойств CRDT отсутствует вероятность конфликта при записи на удалённые узлы.

Также интерес представляет полностью автоматическая синхронизация узлов кластера — это возможно при внедрении динамического выбора узла кластера (например, через выбор случайного узла из кластера на стороне приложения или установки балансировщика нагрузки перед кластером. В таком случае при стабильном потоке запросов, за счёт распределения запросов по узлам узлы автоматически сходятся в пределе к согласованному состоянию, без необходимости поддерживать отдельный поток на каждом узле или отдельный внешний сервис для синхронизации. Схема достижения автоматической синхронизации через использование балансировщика нагрузки представлена на рисунке 7.

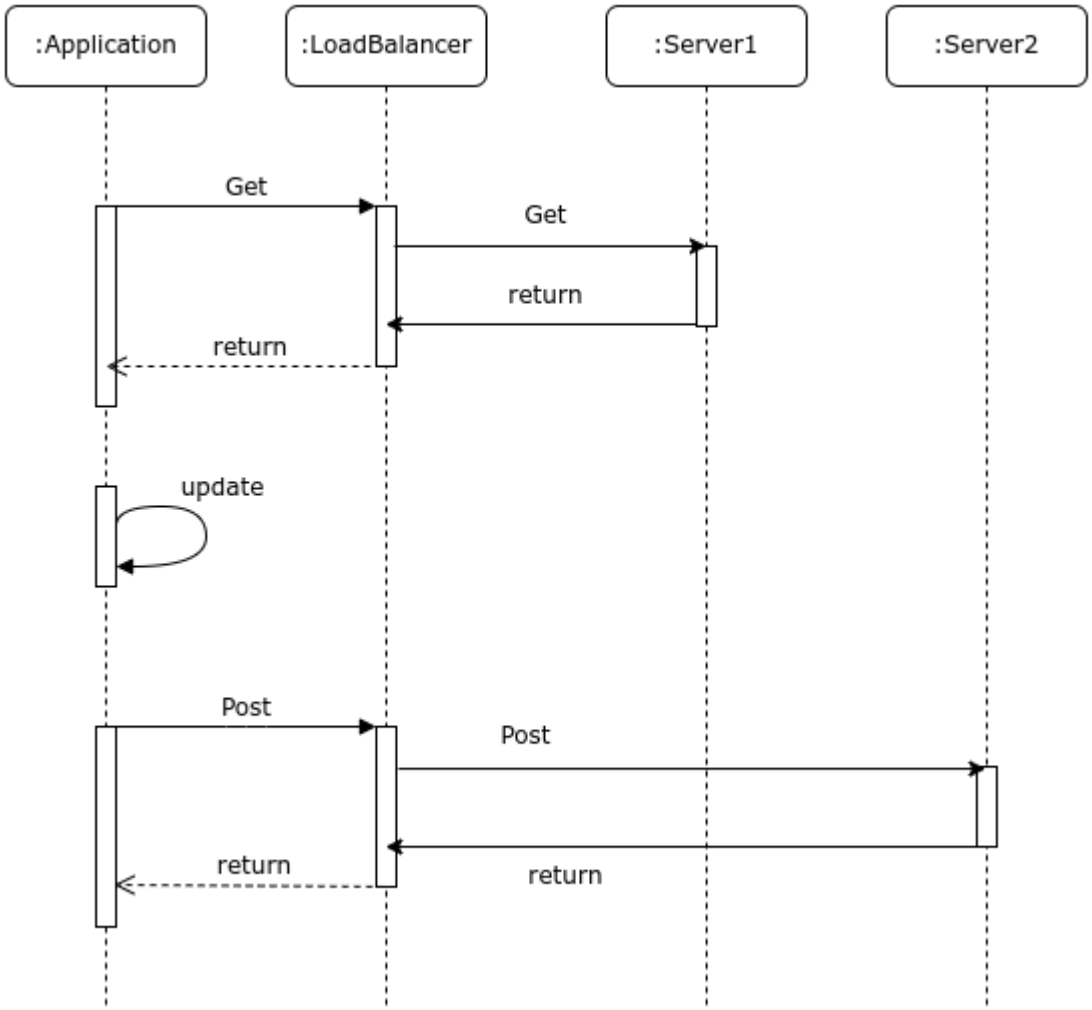

Рисунок 7 – Обеспечение синхронизации данных с использованием балансировщика нагрузки

На первом шаге веб-приложение получает текущее состояние объекта CRDT с произвольного узла кластера на который направит запрос балансировщик нагрузки.

На втором шаге веб-приложение осуществляет модификацию полученного состояния и отправляет изменения в кластер через балансировщик нагрузки. Если балансировщик нагрузки выберет другой узел, например Узел 2, то таким образом с помощью балансировщика нагрузки произойдёт синхронизация узла 1 и узла 2 — узел 2 получит все изменения, которые были на узле 1 в момент когда приложение их прочитало

на шаге 1. При стабильном и постоянном потоке запросов даже при случайном разбрасывании запросов статистически узлы кластера будут сходится к одному состоянию. Даже полностью случайный алгоритм балансировки может быть применён в данном случае, так как свойства алгоритмов CRDT таковы, что каждый узел не чувствителен к получению устаревших или дублирующихся данных. Каждый узел гарантированно может корректно объединить своё состояние с любым другим полученным состоянием. Таким образом, использование балансировщиков нагрузки совместно с алгоритмами CRDT является особенно перспективным методом при практическом внедрении данного подхода к синхронизации данных в веб-приложения.

Таким образом, использование алгоритмов CRDT для решения задачи распределенной синхронизации данных может преодолеть некоторые недостатки существующих подходов за счёт гарантированной сходимости системы к согласованному состоянию с течением времени а также гарантированного отсутствия конфликтов. Также возможно достичь полностью автоматической синхронизации экземпляров приложения между собой за счёт использования балансировщиков нагрузки. Тем не менее, использование алгоритмов CRDT имеет свои ограничения, требующие дополнительного анализа.

1.5 Постановка задачи исследования

По результатам первой главы можно сформулировать задачу исследования. Проблема эффективной синхронизации данных горизонтально масштабированных веб-приложений является актуальной для современных веб-приложений. Существующие подходы к решению этой проблемы обладают недостатками, в то же время альтернативный подход к решению

проблемы (использование алгоритмов CRDT) обладает высоким потенциалом. Таким образом, задачами данного исследования являются:

- Разработать новый подход к разработке и развёртыванию веб-приложений на основе алгоритмов CRDT в соответствии с современными тенденциями разработки веб-приложений;
- Спроектировать практическую реализацию решения проблемы эффективной и надёжной синхронизации данных горизонтально масштабированных веб-приложений с использованием алгоритмов CRDT;
- Проанализировать правильность синхронизации, производительность и нефункциональные характеристики полученной реализации по сравнению с альтернативными реализациями в контексте корпоративных веб-приложений.

Глава 2 Анализ применимости CRDT для решения проблемы эффективной синхронизации данных в горизонтально масштабированных веб-приложениях

2.1 Теоретические основы алгоритмов CRDT

В данном разделе рассматриваются основные теоретические концепции алгоритмов CRDT. Понятие нестрогой согласованности, механизмы автоматического разрешения конфликтов.

Исторически CRDT сильно связаны с так называемыми collaborative systems – системами совместной работы [над документами/файлами] (например Google Docs или MS Office Online). Такие системы позволяют нескольким пользователям работать над одним и тем же документом одновременно. При этом основное желаемое свойство таких систем – конвергентность (сходимость), ситуация, когда после применения всех изменений все клиенты получают одно и то же итоговое состояние документа. Другая важная деталь – система должна сохранять максимально возможное количество внесённых изменений, не предпочитая одни изменения другим и не отбрасывая конфликтующие изменения, чтобы избежать потери пользовательских данных. Для достижения этих целей начиная с 1989 года предлагались различные алгоритмы, относимые к семейству OT (Operational Transformation) [6]. Такие алгоритмы основаны на реплицировании отдельных базовых операций изменения между клиентами системы (в противовес реплицированию полного состояния документа). Однако за годы исследований была доказана некорректность многих предложенных алгоритмов (обнаружены граничные случаи, когда при определённом порядке прихода изменений (например из-за сетевых проблем) система не достигает конвергенции). Отличительная особенность этих

алгоритмов – необходимость центрального сервера, то есть клиенты системы реплицируют свои изменения друг другу используя центральный сервер как посредник. При таком допущении алгоритмы оказываются корректными, однако такое допущение снижает производительность и устойчивость распределенной системы в целом, так как клиенты не могут реплицировать изменения напрямую по модели peer-2-peer (p2p). Алгоритмы CRDT представляют собой развитие идей OT и призваны преодолеть их недостатки (трудность разработки корректного алгоритма и необходимость центрального сервера). CRDT позволяют более децентрализованный подход когда узлы могут реплицировать данные напрямую между собой, без участия центрального сервера. Основные различия алгоритмов OT и алгоритмов CRDT представлены в таблице 3.

Таблица 3 – Основные особенности алгоритмов Operational Transformation (OT) и алгоритмов CRDT

Характеристика	Алгоритмы Operational Transformation (OT)	Алгоритмы CRDT
Сложность реализации	Высокая	Средняя
Теоретическая обоснованность	Ограниченная	Да
Возможность синхронизации без центрального сервера	Нет	Да

Чтобы понять, как именно CRDT помогают в распределенной синхронизации данных следует сформулировать решаемую ими задачу. В связи с этим полезно вспомнить знаменитую CAP-теорему – эмпирическое наблюдение о том, что в любой распределенной системе возможно достичь лишь двух из трёх полезных свойств (Consistency – целостность данных (обычно под этим словом понимается мгновенное отражение изменений в

данных, т.е. после внесения изменений изменения сразу оказываются на всех узлах системы, все чтения после момента изменения сразу читают обновлённые данные), Availability – доступность (под этим словом обычно подразумевается то, что система корректно отвечает на 100% запросов) и Partition Tolerance – устойчивость к разделению (при разделении распределенной системы на несколько изолированных частей каждая из частей продолжает выдавать корректные ответы, т.е. система устойчива к сетевым проблемам и возможному выходу из строя некоторых узлов). В этой парадигме CRDT подходит к построению распределенной системы через ослабление свойства Consistency. Задача построения корректной распределенной системы переформулируется в построение системы, обладающей свойством согласованности в конечном счёте (Strong Eventual Consistency). Этим словосочетанием называется свойство системы “как только все узлы системы обменяются своими изменениями, каждый из узлов придёт к одному и тому же состоянию, без потерь данных”. Слово “strong” отражает тот факт, что для достижения итоговой согласованности узлам распределенной системы не обязательно поддерживать определённый порядок применения обновлений из-за чего отсутствуют конфликты при репликации и не требуется вмешательство пользователя для их разрешения. При этом одинаковость состояния в каждый конкретный момент времени не гарантируется, то есть узлы системы могут иметь различные состояния в процессе обмена данными, постулируется только итоговая сходимость. Именно благодаря такому переформулированию задачи становится возможным избежать использования распределенных блокировок и обязательного кворума. Однако при этом возможно чтение устаревших данных (stale reads, чтения, которые происходят до достижения полной согласованности). При этом другие два свойства (Availability и Partition

Tolerance) продолжают выполняться полностью (таблица 4). Получившаяся распределенная система с такими свойствами может оказаться приемлемой для многих практических веб-приложений (но не для всех).

Таблица 4 – Основные свойства алгоритмов CRDT с точки зрения CAP-теоремы

Свойство	Выполняется	Пояснение
Consistency (Согласованность)	Нет	Полная согласованность данных. Зафиксированные данные мгновенно оказываются доступными для всех экземпляров приложения. Невозможна повреждение данных в результате одновременной записи. Алгоритмы CRDT не обеспечивают полную согласованность данных в любой момент времени, лишь гарантируют достижение согласованности с течением времени.
Availability (Доступность)	Да	Система корректно отвечает на 100% запросов.
Partition Tolerance (Устойчивость к разделениям)	Да	При разделении распределенной системы на несколько изолированных частей каждая из частей продолжает выдавать корректные ответы, т.е. система устойчива к сетевым проблемам и возможному выходу из строя некоторых узлов.

CRDT позволяют построить распределенную систему с согласованностью в конечном счёте (Strong Eventual Consistency). Особенностью CRDT является отсутствие конфликтов при репликации изменений по узлам системы (conflict-free). Если два CRDT-объекта получили один и тот же набор изменений, то они алгоритмически достигнут одного и того же состояния путём применения математических правил,

которые гарантируют сходимость состояний (без вмешательства пользователя для разрешения конфликтов) [27]. Объект CRDT может быть изменён на любом из узлов системы без координации с другими узлами (это позволяет системе в целом переживать временные отключения узлов и автоматически сходится к консистентному состоянию как только соединение с отключёнными узлами восстановится). Кроме того, CRDT устойчивы к возможному дублированию изменений, то есть не требуют exactly once доставки изменений, достаточно at least once. Это позволяет упростить разработку каналов репликации между узлами и повысить производительность системы (нет необходимости обеспечивать уникальность изменений). По сути один и тот же набор изменений приводит к одному и тому же результату на каждом из узлов, независимо от порядка их выполнения или дупликации на конкретных узлах распределенной системы.

Основные полезные свойства алгоритмов CRDT перечислены в таблице 5.

Таблица 5 – Основные полезные свойства алгоритмов CRDT

Свойство	Пояснение
Отсутствие конфликтов при синхронизации данных	Любые два узла гарантированно могут объединить свои состояния. Неопределённость результата не возникает и не требуется вмешательство пользователя для ручного разрешения конфликта.
Отсутствие «потерянных» изменений	Изменение, внесённое на одном узле не может пропасть в процессе синхронизации (кроме случаев когда оно было отменено позднее).
Гарантированная сходимость узлов системы к одному состоянию	Как только все узлы смогут обменятся изменениями система гарантированно приходит к единому состоянию
Возможность вносить изменения даже на временно изолированных узлах	Даже изолированные узлы могут корректно принимать запросы на изменение

Продолжение таблицы 5

Свойство	Пояснение
Отсутствие необходимости строгой упорядоченности изменений	Изменения могут реплицироваться по узлам системы в произвольном порядке, независимо от их хронологии
Отсутствие необходимости поддерживать уникальность изменений, которыми обмениваются узлы системы	Узлы системы могут обмениваться изменениями с другими узлами, не заботясь о том, получал ли принимающий узел данное изменение ранее

Конкретный CRDT это специально разработанный тип данных с определёнными свойствами. А именно, должны выполнятся следующие математические выражения:

- Операции над типом данных ассоциативны ($(x \circ y) \circ z = z \circ (y \circ z)$, т.е. порядок группировки операций не имеет значения);
- Операции над типом данных коммунитативны ($x \circ y = y \circ x$), т.е. порядок операндов операции не имеет значения);
- Операции идемпотенты ($x \circ y \circ y = x \circ y$), т.е. при повторении применении операции результат не изменяется).

Эти три свойства математически формализуют особенности CRDT и позволяют им обеспечивать согласованность в конечном счёте (Strong Eventual Consistency) и эффективную и устойчивую репликацию внутри распределенной системы.

Разработка конкретного CRDT таким образом заключается в создании типа данных и операций над ним, которые согласуются с бизнес-логикой конкретного веб-приложения и его предметной областью и удовлетворяют трём вышеупомянутым свойствам.

Существует три основных вида CRDT [23], рассмотрим их:

- CvRDT (ConVergent или state-based),

- CmRDT (CoMmutative или op-based),
- delta-based CRDT.

CvRDT (ConVergent или state-based) – CRDT с хранением состояния, при изменении объекта CRDT для его репликации на соседние узлы отправляется весь изменённый объект целиком. Для корректной репликации необходимо определить функцию слияния двух состояний. Каждая реплика будет применять эту функцию к своему состоянию и полученному с другой реплики обновлённому состоянию. Пример содержимого запросов на репликацию в формате JSON, которыми могут обмениваться узлы, хранящие ассоциативные массивы представлены в таблице 6.

Таблица 6 – Пример запросов на репликацию состояния ассоциативного массива для CvRDT

Номер запроса	Содержимое запроса (JSON)
1	{"key": {"value": "x", "timestamp": 1}, "key2": {"value": "y", "timestamp": 2}}
2	{"key": {"value": "x", "timestamp": 3} "key2": {"value": "z", "timestamp": 4} "key3": {"value": "m", "timestamp": 4}}

CmRDT (CoMmutative или op-based) – коммутативные CRDT, при изменении объекта CRDT для его репликации на соседние узлы отправляется только конкретная операция изменения. Примеры запросов на репликацию в формате JSON для данного вида представлены в таблице 7. Оба вида CRDT в итоге приводят к одному и тому же результату, один вид можно реализовать через другой (т.е. вычислять операции на основе разницы состояния или наоборот конструировать новое состояние путем применения операции).

Таблица 7 – Пример запросов на репликацию состояния ассоциативного массива для CmRDT

Номер запроса	Содержимое запроса (JSON)
1	{"operation": "add", "key": "key1", "value": "x", "timestamp": 1}
2	{"operation": "modify", "key": "key2", "value": "z", "timestamp": 2}

delta-based CRDT является промежуточным между первыми двумя. В данном виде CRDT проводится агрегация операций изменения перед рассылкой на другие узлы. Можно рассматривать этот вид CRDT как оптимизацию CmRDT. Примеры запросов на репликацию в формате JSON для данного вида представлены в таблице 8.

Таблица 8 – Пример запросов на репликацию состояния ассоциативного массива для delta-based CRDT

Номер запроса	Содержимое запроса (JSON)
1	{"operation": "add", "key": "key1", "value": "x", "timestamp": 1, "operation": "modify", "key": "key2", "value": "z", "timestamp": 2}
2	{"operation": "modify", "key": "key2", "value": "k", "timestamp": 3, "operation": "modify", "key": "key3", "value": "m", "timestamp": 3}

Конкретные реализации алгоритмов CRDT обычно используют массивы значений, где каждый экземпляр приложения изменяет только своё значение. Также для управления конфликтами могут использоваться временные метки. Подробнее конкретные алгоритмы CRDT рассмотрены в разделе 2.4. Рассмотрим для примера одну из реализаций множества, не использующую временные метки - PN-Set. В данном алгоритме каждому элементу ставится в соответствие счетчик, при добавлении элемента в

множество счетчик увеличивается, при удалении – уменьшается. Состояние двух реплик в таком случае объединяется путем складывания счетчиков элементов. Схема работы данной реализации приведена на рисунке 8. Другие реализации множества позволяют обойтись без создания временных идентификаторов через введение приоритета операций (например, операция добавления более приоритетна, чем операция удаления или наоборот). Выбор конкретной реализации зависит от желаемой семантики структуры данных. Существуют различные CRDT-реализации для списка, словаря, графа и других структур данных (в том числе вложенных).

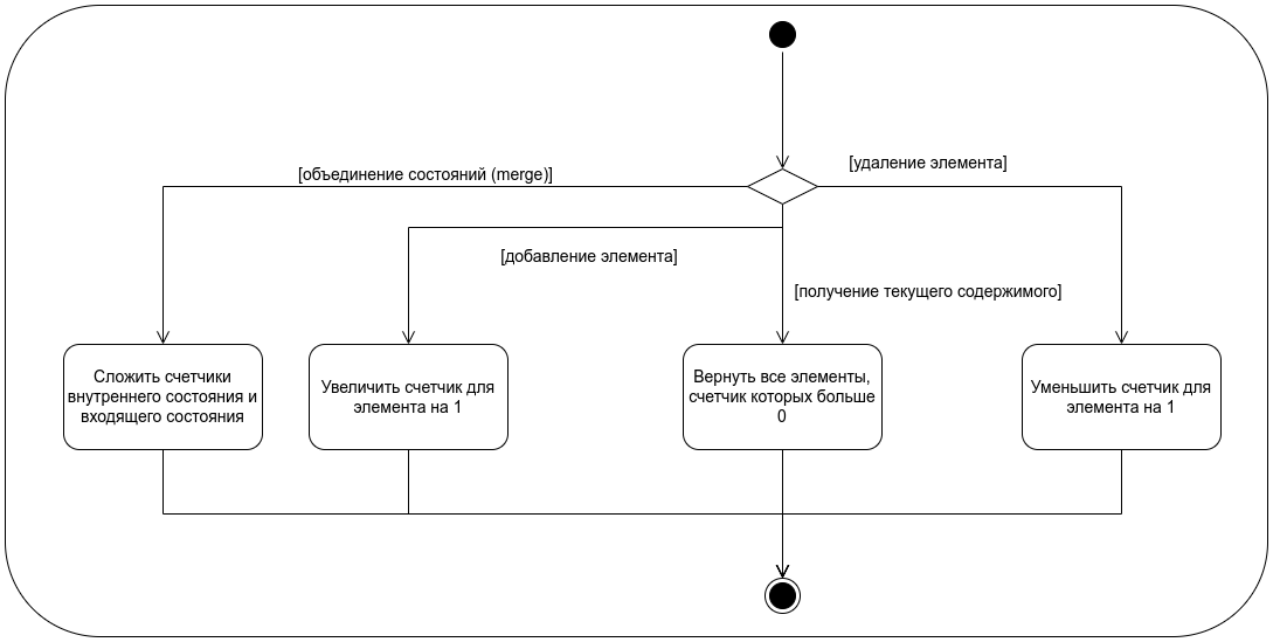

Рисунок 8 – Алгоритм работы PN-Set

Таким образом, рассмотрены теоретические основы алгоритмов CRDT и подход CRDT в сравнении с другими методами построения распределенных систем. Особенность алгоритмов CRDT — использование ослабленных требований к согласованности данных для достижения преимуществ при реализации других требований (доступность, отсутствие конфликтов, простота реализации). Рассмотрена также конкретная структура

множества PN-Set, реализующая свойства CRDT. Конкретные реализации основаны на использовании счётчиков, массивов, где у каждого узла системы имеется своё значение, а также на использовании временных меток и меток удаления.

2.2 Текущее применение алгоритмов CRDT

CRDT имеет несколько применений на практике. Основные области применения перечислены в таблице 9.

Таблица 9 – Основные сферы применения алгоритмов CRDT

Сфера применения	Количество применений
Специализированные распределенные базы данных	3
Программы для совместного редактирования документов	3
Исследовательские проекты	2

Первая из областей применения – специализированные распределенные базы данных. Например, CRDT используется в Redis (Enterprise-версия) для гео-репликации [1] и Riak [11]. Обе упомянутые БД принадлежат типу “ключ-значение” (key-value). Кроме того, CRDT как один из методов разрешения конфликтов поддерживается в облачной Microsoft Azure CosmosDB [7].

Если говорить о более узких применениях, то CRDT применяются в нескольких коллаборативных текстовых редакторах/системах создания заметок(Teletype [37], PeerPad [28]), а также в популярном коллаборативном приложении для веб-дизайна Figma [14] (хотя и с использованием центрального сервера). CRDT также используются в онлайн-игре League of Legends[24], навигационной системе TomTom (для синхронизации

пользовательских данных по нескольким устройствам) [9], стриминговом сервисе SoundCloud для отображения ленты событий пользователя [30] и в крупнейшем онлайн-маркетплейсе в мире Amazon (для синхронизации корзины) [18].

Кроме того, существует несколько библиотек и SDK, которые ставят целью сделать CRDT применимыми в множестве прикладных веб-приложений (без привязки к конкретной предметной области). Например JavaScript-библиотека Yjs ставит своей целью предоставлять интерфейс стандартных JavaScript-типов array и map, изменения которых автоматически и бесконфликтно реплицируются по всем узлам сети, в манере p2p [41]. Одно из новейших достижений в сфере популяризации CRDT – создание библиотеки Automerge. Эта библиотека стремится предоставить максимально совместимое с распространенным форматом JSON API, но с поддержкой бесконфликтного внесения одновременных изменений в объекты несколькими пользователями [3]. При этом поддерживается работа без центрального сервера, а сетевое взаимодействие абстрагируется и может быть реализовано с использованием нескольких протоколов (например WebRTC, TCP, WebSocket).

Несмотря на значительный прогресс в новейших исследованиях CRDT, CRDT все ещё не так широко применимы, как они могли бы быть. CRDT применяются не массово, не для широкого круга веб-приложений в различных предметных областях, а лишь в крупных компаниях, когда они встречаются с проблемами масштабирования и реализуют CRDT для своих конкретных задач. Наблюдается недостаток открытых общих решений, которые бы упростили массовое внедрение CRDT в веб-приложения. Существующие решения по историческим причинам развития CRDT часто привязаны к одной предметной области - коллаборативные системы

редактирования документов и обращают меньшее внимание на другие задачи. Кроме того, многие библиотеки фокусируются на p2p модели и прямом взаимодействии клиентов, используют выполнение JavaScript-кода на клиенте и не предоставляют абстракции для клиент-серверной модели веб-приложения (её серверной части). Также часто отсутствуют привязки (bindings) к компилируемым языкам, которые чаще используются для серверной части веб-приложений, чем JavaScript.

Таким образом на основе проведённого анализа, на данный момент CRDT применяются в веб-приложениях ограниченно. В основном они используются для решения специализированных задач крупных корпораций. Широкого внедрения CRDT в массовые корпоративные веб-приложения не наблюдается.

2.3 Выбор предметной области использования веб-приложения для внедрения алгоритмов CRDT

CRDT представляют собой семейство алгоритмов, а не одну конкретную реализацию. Каждый конкретный алгоритм семейства CRDT обладает своей спецификой и может быть применим лишь в некотором подмножестве веб-приложений. То есть невозможно отыскать один универсальный алгоритм для всех веб-приложений. Например, корзина покупателя интернет-магазина хорошо моделируется через структуру данных «множество», а для решения проблем распределенной синхронизации подходят алгоритмы слияния множеств с обеспечением свойств CRDT. В то же время банковские транзакции покупателя плохо моделируются подобными алгоритмами, так как в сфере банковских услуг чрезвычайно важно обеспечивать строгий и однозначный порядок транзакций, а также существуют повышенные требования к безопасности и согласованности данных. В подобных областях знаний алгоритмы CRDT принципиально

слабо применимы из-за своей не строгой согласованности. В таблице 10 представлена информация о применимости алгоритмов CRDT в различных видах веб-приложениях.

Таблица 10 – Применимость алгоритмов CRDT в различных веб-приложениях

Вид веб-приложений	Примеры веб-приложений	Применимость CRDT
Веб-приложения, требовательные к строгой согласованности данных	Веб-приложения, связанные с финансовыми транзакциями, страховой деятельностью, веб-приложения для корпоративных клиентов	слабая
Веб-приложения, нейтральные к строгой согласованности данных	Веб-приложения связанные с потреблением развлекательного контента, сетевые игры	средняя
Веб-приложения, способные извлекать прямую выгоду из не строгой согласованности	Веб-приложения для мониторинга информационных систем, построения отчетов, интернет-магазины	высокая

Таким образом, алгоритмы CRDT слабо применимы в тех веб-приложениях, которым недостаточно нестрогой согласованности (Eventual Consistency), а нужна строгая согласованность (Strong Consistency) (например, требуется отсутствие возможности прочитать устаревшие данные). Простейший пример – аукцион, в общем случае нельзя допускать чтобы данные о победителе были разные на разных узлах, победитель должен быть один во всей системе. Целый класс платёжных веб-приложений также скорее всего не сможет воспользоваться преимуществами CRDT так как одна из основных задач таких веб-приложений – недопущение “двойных трат” (трата уже потраченных средств), которые могут случиться при нестрогой согласованности и временной рассинхронизации узлов

распределенной системы. Оценить применимость алгоритмов CRDT можно путём анализа последствий временной несогласованности данных. Результаты проведённого анализа представлены в таблице 11.

Таблица 11 – Оценка возможных последствий временной несогласованности данных

Последствия временной несогласованности данных	Пример	Оценка последствий	Применимость алгоритмов CRDT
Репутационные потери	Рассогласованность данных в банке, торговой площадке, посреднической платформе, страховой компании, брокере, медицинской организации	Негативные	Слабая
Финансовые потери	Рассогласованность данных в банке, торговой площадке, посреднической платформе, страховой компании, рекламной бирже	Негативные	Слабая
Незначительная неактуальность данных	Рассогласованность приблизительных данных (например, счетчика просмотров, вспомогательной агрегированной витрины отчётов)	Нейтральные	Средняя
Незначительная неактуальность данных и возможная финансовая выгода	Интернет-магазины (например, сохранять в корзине даже товары, добавленные с отключившихся устройств)	Положительные	Высокая

Алгоритмы CRDT наиболее применимы в веб-приложениях интернет-магазинов. В подобных веб-приложениях чрезвычайно важно сохранять заказы пользователей даже при нестабильном соединении (это напрямую влияет на выручку компании-оператора интернет-магазина), и в то же время может быть допустима нестрогая согласованность данных.

Таким образом, алгоритмы CRDT изначально применимы только в подмножестве веб-приложений. Для применимости необходимо, чтобы с точки зрения приложения была допустима временная неактуальность данных (нестрогая согласованность). Наиболее перспективным выглядит внедрение алгоритмов CRDT в веб-приложения интернет-магазинов.

2.4 Постановка задачи обеспечения работы корзины товаров

Необходимо реализовать виртуальную корзину товаров для веб-приложения интернет-магазина.

Корзина товаров в интернет-магазине — виртуальный аналог физической корзины товаров. Корзина товаров служит как хранилище выбранных пользователем товаров до момента оформления заказа. Наличие виртуальной корзины товаров в пользовательском интерфейсе обеспечивает наличие единой точки просмотра выбранных товаров, также в корзине часто подсчитывается текущая стоимость заказа. Кроме того, наличие корзины как элемента пользовательского интерфейса позволяет отложить момент длительного оформления заказа (заполнение большого числа параметров доставки и оплаты, возможно регистрация) и повысить конверсию.

Полученная реализация должна реализовывать прецеденты, представленные на рисунке 9.

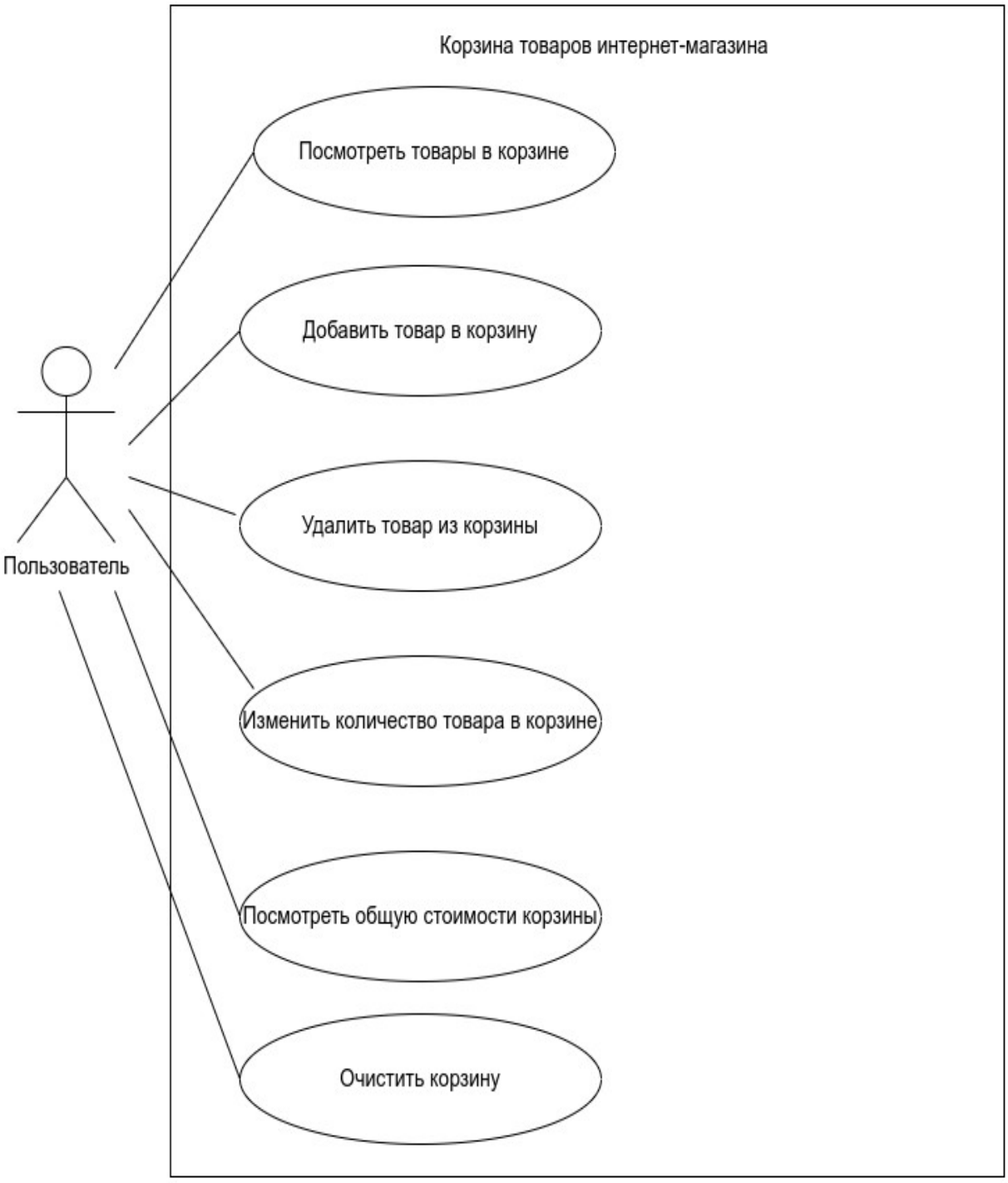

Рисунок 9 – Диаграмма прецедентов корзины товаров интернет-магазина

Подробная спецификация представлена в таблице 12.

Таблица 12 – Основные операции корзины товаров интернет-магазина

Операция	Пояснение
Отображение товаров в корзине (get)	Отображается текущее состояние корзины. Товары должны отображаться вместе с их ценами и количеством. Удалённые товары не должны отображаться.
Добавление товара в корзину (add)	Товар добавляется в корзину. Должна быть возможность добавить несколько одинаковых товаров. Корзина также должна запоминать цену товара для последующего отображения.
Удаление товара из корзины (remove)	Товар удаляется из корзины и перестаёт учитываться при подсчёте суммы заказа.
Изменение товара в корзине (set)	Изменяется свойства добавленного в корзину товара. Например, количество выбранных товаров, соответствующим образом должна изменится сумма заказа.
Подсчёт общей стоимости заказы (sum)	Подсчитывается общая сумма корзины. При этом должно учитываться количество товаров и не учитываться удалённые.
Очистка корзины (clean)	Содержимое корзины отчищается, после данной операции корзина не должна содержать ни одного товара.

Следует отметить, что действительно необходимыми являются только 3 операции — получения содержимого корзины, добавления товаров в корзину и удаления товара из корзины. Остальные операции могут быть реализованы через эти две базовые операции. Например, изменение количества товаров может быть реализовано через удаление и последующе добавление этого же товара с другим количеством, подсчёт общей стоимости через отображение корзины и суммирование стоимостей всех товаров снаружи корзины, а очистка корзины через получение содержимого корзины и проведение операции удаления товара для каждого из полученных товаров. Кроме необходимых операций существуют нефункциональные требования к реализации корзины (представлены в таблице 13).

Таблица 13 – Нефункциональные требования к реализации корзины

Требование	Пояснение
Приемлемый уровень производительности (не более одной секунды на операцию)	Высокая производительность позволяет отображать более актуальное состояние и иметь более отзывчивый пользовательский интерфейс
Отсутствие «потерянных» товаров	Однажды добавленный в корзину товар не должен пропадать из корзины до удаления/очистки. Данное свойство позволяет избежать потери пользовательских данных а также увеличить продажи.
Стандартный формат обмена данными (JSON или XML)	Использование стандартизованного формата позволяет легче подключать реализацию корзины в различное клиентское программное обеспечение (браузерный код, мобильные приложения, различные интеграции).

Таким образом осуществлена постановка задачи на практическую реализацию синхронизации корзины товаров интернет магазина с использованием алгоритмов CRDT. Разработаны требования к корректной реализации корзины.

2.5 Выбор алгоритма CRDT для внедрения в веб-приложение интернет-магазина

Для выбора подходящего алгоритма были проанализированы 3 известных алгоритма CRDT. Для анализа использовались следующие критерии:

1. Выполнимость вариантов использования (0-6). В рамках данного критерия алгоритму присваивается по одному баллу за возможность реализации каждого из шести вариантов использования (Отображение товаров в корзине (get), Добавление товара в корзину (add), Удаление

товара из корзины (remove), Изменение количества товаров одного вида, Подсчёт общей стоимости заказы, Очистка корзины).

2. Выполнимость нефункциональных требований к корзине товаров интернет-магазина (0-3). В рамках данного критерия присваивается по одному баллу за возможность выполнения каждого из трех нефункциональных требований (Приемлемый уровень производительности, Отсутствие «потерянных» товаров, Стандартный формат обмена данными).

Рассмотрим основные алгоритмы CRDT, применяя выявленные критерии.

Observed-Removed Set (OR Set). Эта реализация CRDT соответствует семантике множества, относительно проста и при этом избегает некоторых недоработок более ранних реализаций (например, в некоторых ранних реализациях удаленный элемент невозможно повторно добавить в множество) [17]. OR Set также отдает приоритет операции добавления при параллельном добавлении и удалении.

Внутренняя реализация использует два множества — множество добавленных элементов и множество удалённых элементов. При этом каждому элементу в любом из двух множеств ставится в соответствие уникальная временная метка. Операция удаления добавляет элемент в множество удалённых элементов, операция добавления – в множество добавленных. При чтении элементов множества каждая реплика выводит содержимое своего множества добавленных элементов, при этом удаляя элементы, которые находятся в множестве удалённых и обладают более новой временной меткой чем этот же элемент в множестве добавленных элементов. При такой реализации поддерживается удаление и повторное добавление элемента, которое становится возможным благодаря упорядоченности операций по времени. Каждая реплика может объединить

своё состояние с другой репликой путём объединения обеих множеств. Данный алгоритм отражён на рисунке 10.

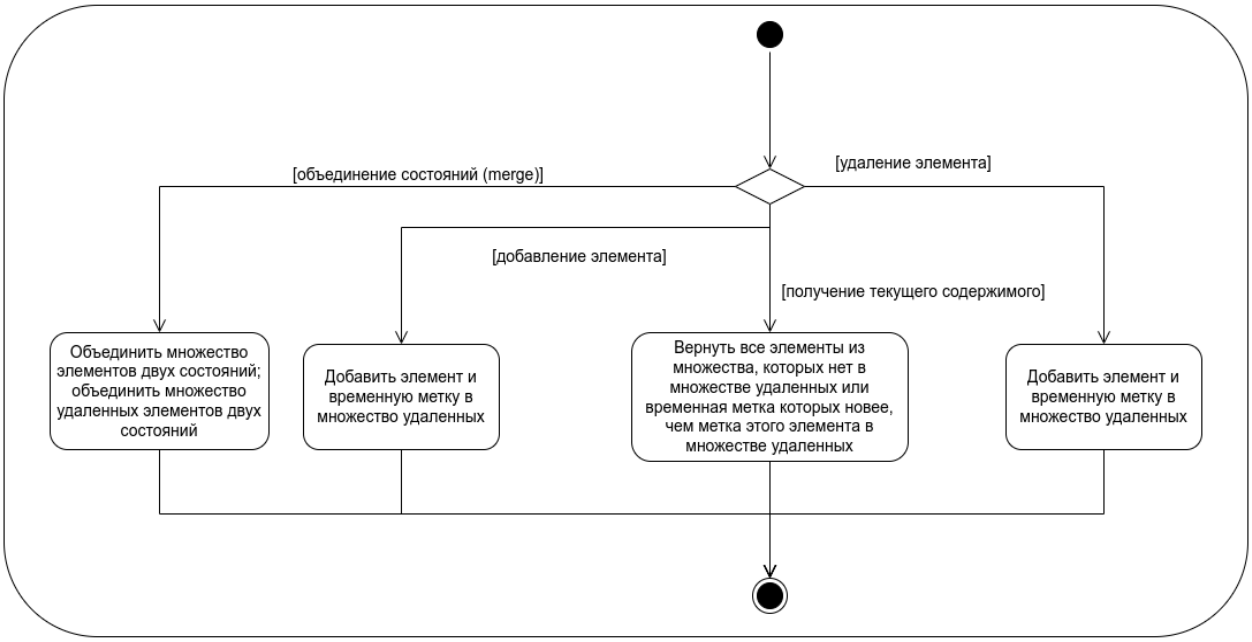

Рисунок 10 – Алгоритм CRDT OR-Set

С помощью OR-Set возможно хорошо смоделировать корзину товаров интернет-магазина как множество товаров. Каждый товар хранит идентификатор, название, цену и количество. При такой реализации возможно реализовать все варианты использования корзины товаров, а также нефункциональные требования и поэтому данный алгоритм получает максимальную оценку применимости.

Increment-Only Counter (G-Counter). Данный алгоритм CRDT представляет собой монотонно возрастающий счетчик. Счетчик поддерживает только одну операцию изменения — инкрементирование (увеличение на единицу), а также операцию чтения значения. Счетчик на самом деле представляет собой массив размера N (количество узлов в распределенной системе). Каждая реплика при увеличении счетчика увеличивает значение элемента массива, соответствующее своему номеру.

При репликации каждая реплика объединяет своё состояние с пришедшим состоянием путем взятия максимума по каждому из элементов двух массивов состояний. Работа данного алгоритма отражена на рисунке 11. Не трудно показать, что такая реализация CRDT позволяет достичь согласованности в конечном счёте (Strong Eventual Consistency), при этом порядок репликации обновлений не имеет значения. Задача поддержки понижения значения счётчика решается через введение второго массива, который хранит уменьшения.

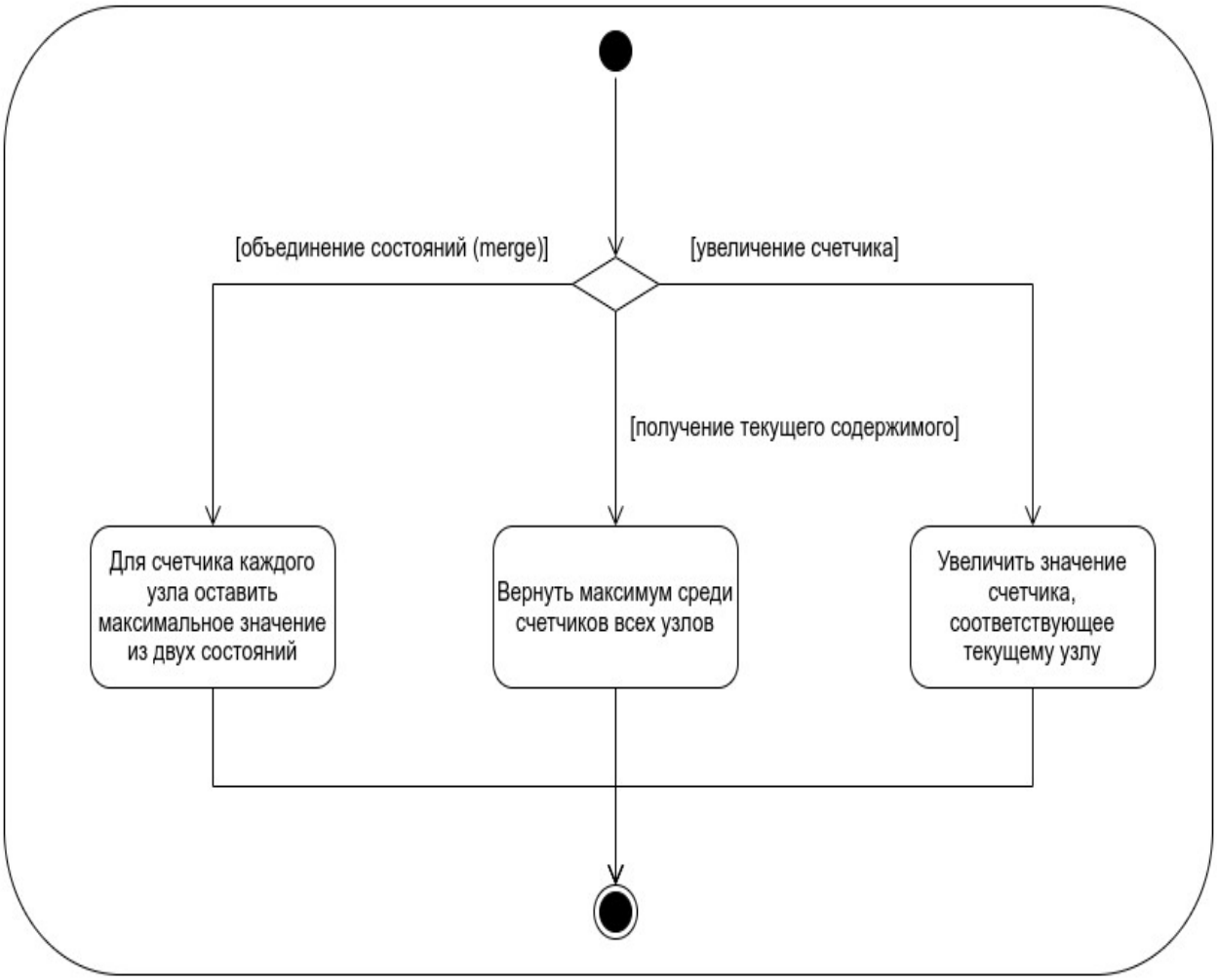

Рисунок 11 – Алгоритм Increment only counter

Для реализации корзины товаров интернет-магазина необходимо модифицировать данный алгоритм для поддержки операции уменьшения значения. Однако даже в этом случае, реализация корзины товаров на основе данного алгоритма будет поддерживать только один товар. Для выполнения всех вариантов использования корзины необходимо разработать более сложный алгоритм с несколькими счетчиками и даже в этом случае остается не реализованным прецедент рассчета стоимости корзины — негде хранить цены добавленных товаров. Поэтому данный алгоритм получает пониженную оценку применимости.

Last-Write-Wins Register (LWW-Register). Данный алгоритм реализует ячейку памяти, которая поддерживает операцию чтения и операцию записи. При этом более новые операции записи имеют приоритет. Для этого каждой операции записи присваивается уникальный идентификатор, который можно упорядочить по времени (например, timestamp). После этого каждая реплика может объединить своё состояние с состоянием другой реплики через сравнение идентификаторов. Более новый идентификатор побеждает и определяет результирующее состояние. Само содержимое регистра может быть любого типа. Работа алгоритма отражена на рисунке 12.

С помощью данного алгоритма возможно реализовать корзину товаров с поддержкой всех вариантов использования. Однако, в рамках данного алгоритма невозможно реализовать нефункциональное требование «отсутствие потерь пользовательских данных».

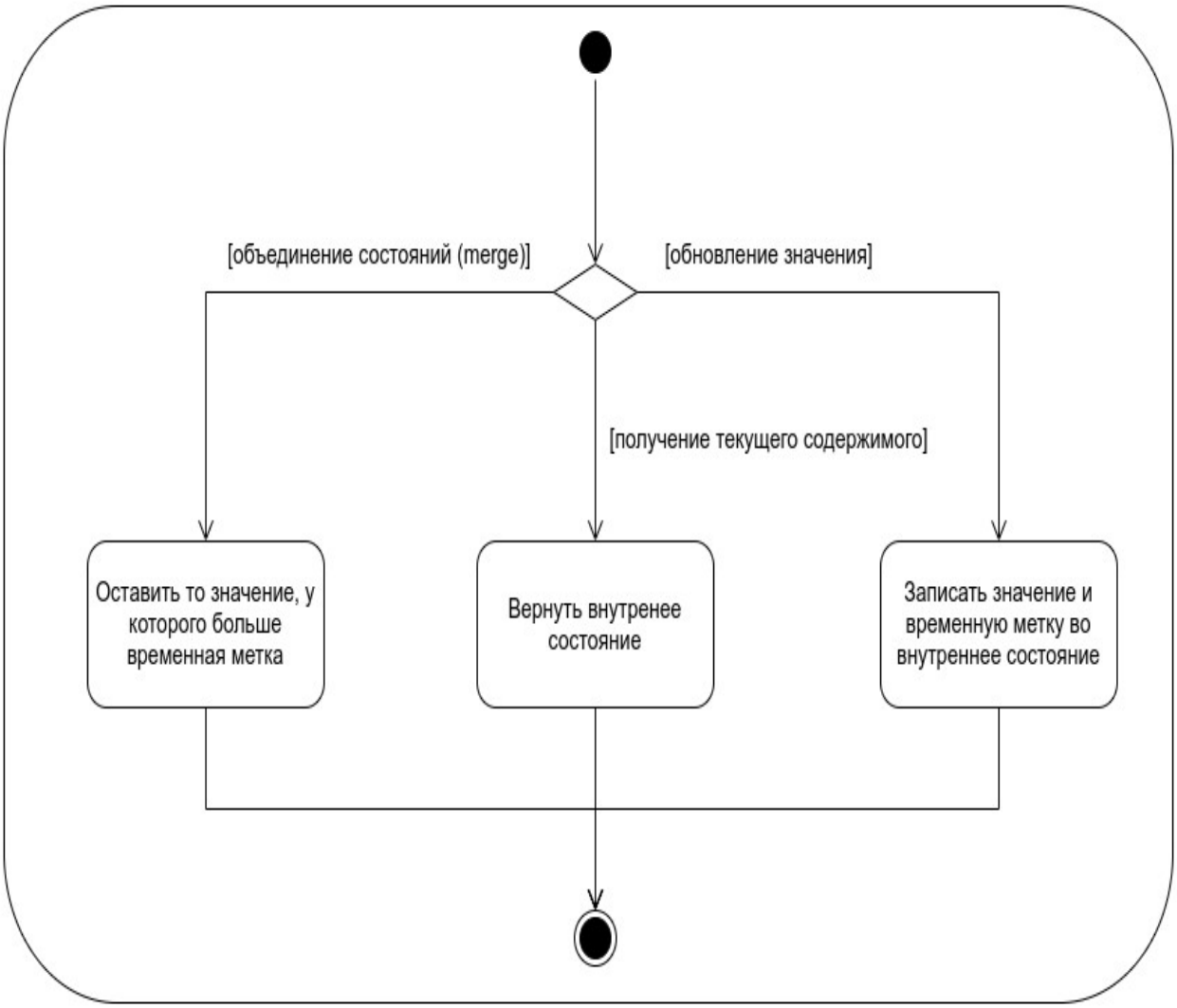

Рисунок 12 – Алгоритм Last-Write-Wins Register

При наличии нескольких пользовательских сессий и временных обрывов соединения возможна ситуация, при которой часть обновлений корзины, сделанных во время отсутствия соединения пропадет. Такая ситуация возникает так как данный алгоритм полностью заменяет состояние на более новое, без более продвинутого объединения двух корзин. По этой причине данный алгоритм получает пониженную оценку применимости.

Результаты анализа приведены в таблице 14.

Таблица 14 – Результаты классификации веб-приложений

Алгоритм CRDT	Выполнимость вариантов использования (0-6)	Выполнимость нефункциональных требований (0-3)	Итого
Observed-Removed Set (OR Set) [4]	6	3	9
Increment-Only Counter (G-Counter) [5] [35]	5	3	8
Last-Writer-Wins Register (LWW-Register) [33]	6	2	8

Таким образом, на основе анализа для реализации корзины товаров товаров интернет-магазина был выбран алгоритм Observed-Removed Set (OR Set), представленный на рисунке 10. Данный алгоритм позволяет реализовать все выявленные варианты использования корзины, а также нефункциональные требования к корзине.

2.6 Постановка задачи на реализацию синхронизации данных с использованием алгоритма CRDT «OR Set»

Необходимо реализовать алгоритм CRDT Observed-Removed Set (OR Set), рассмотренный в 2.5. Таким образом, задачи на реализацию синхронизации данных в горизонтально масштабированных веб-приложениях на примере корзины товаров интернет-магазина формулируются следующим образом:

- Спроектировать и реализовать корзину товаров интернет-магазина с поддержкой синхронизации данных на основе алгоритма CRDT «OR Set»;

- Проверить правильность синхронизации корзины товаров в условиях сетевых проблем;
- Реализовать корзину товаров интернет-магазина с использованием различных альтернативных технологий и сравнить с полученной реализацией на основе алгоритма CRDT «OR Set».

Для сравнения различных реализаций корзин товаров можно воспользоваться критериями, представленными в таблице 15.

Таблица 15 – критерии сравнения различных реализаций корзин

Критерий	Пояснение	Важность критерия (1-5)
Корректность	Корзина должна хранить и отображать все когда-либо добавленные добавленные товары и не отображать удалённые товары, корректно рассчитывать сумму покупок.	5
Производительность	Реализацию корзины должна минимизировать задержу выполнения операций с корзиной	3
Количество ошибок при репликации	Реализация корзины, должна минимизировать количество ошибок при репликации	2

Глава 3 Проектирование и реализация решения проблемы синхронизации данных горизонтально масштабированных веб-приложений с использованием алгоритмов CRDT на примере веб-приложения интернет-магазина

Приложения интернет-магазины являются одними из наиболее подходящих типов веб-приложений для внедрения алгоритмов CRDT. В данной главе осуществляется проектирование и реализация конкретного алгоритма CRDT (OR Set) для веб-приложения интернет-магазина. Полученная реализация должна решать проблему синхронизации данных горизонтально масштабированных веб-приложений на конкретном примере синхронизации корзины товаров интернет-магазина.

3.1 Разработка алгоритма эффективной синхронизации корзины товаров с использованием алгоритма CRDT OR Set

Рассмотрим типичное веб-приложение интернет-магазина. По критериям практической распространённости рассматриваем монолитное веб-приложение с использованием Базы Данных и без использования облачных услуг. В соответствии с проведённым в предыдущих разделах анализом, для данного типа веб-приложения и задачи обеспечения синхронизации корзины товаров наиболее подходящей реализацией CRDT является Observed-Removed Set (OR-Set) в виде соответствующих модулей (клиентского и серверного). В данном случае под синхронизацией понимается приведение состояния N клиентов и M серверов к единой версии. Состояние хранится в оперативной памяти (in-memory), а для резервного копирования может периодически сохранятся в реляционную БД. Тем не менее, в данной системе реляционная СУБД играет вспомогательную роль и

при нормальной работе системы используется исключительно состояние в памяти.

Использовался алгоритм Observed-Removed Set, реализованный на языке программирования Python. Полученная клиентская часть веб-приложения хранит корзину товаров в оперативной памяти и обращается на серверную часть веб-приложения по протоколу HTTP при необходимости внесения изменений. При получении изменений серверная часть веб-приложения объединяет полученное состояние со своим внутренним состоянием в оперативной памяти и отправляет результат слияния состояний в ответе. При получении ответа клиентская часть также осуществляет объединение чтобы учесть возможные новые изменения.

Для более полного понимания архитектуры развёртывания рассмотрим полный «путь» синхронизации корзины пользователя от внесённого пользователем веб-приложения изменения в браузере до синхронизации состояния корзины на сервере.

На первом шаге пользователь добавляет или удаляет элемент из корзины через браузерный интерфейс. Браузер регистрирует это событие и передаёт управление обработчику. Обработчик события (модуль CRDT) локально сохраняет изменения и инициирует синхронизацию корзины с сервером через отправку HTTP-запроса. В теле запроса отправляется текущее состояние корзины пользователя. Запрос проходит через прокси и балансировщик нагрузки и доходит до серверной части веб-приложения. Серверная часть веб-приложения обрабатывает запрос с помощью серверного модуля CRDT, полученное от клиента состояние корзины объединяется с текущим серверным состоянием (merge) и сохраняется в Базу Данных для обеспечения персистентности данных. Затем сервер возвращает полученное после объединения итоговое состояние корзины в теле HTTP

ответа. Ответ проходит прокси-сервера в обратном порядке и доходит до отправителя. Клиентский модуль CRDT объединяет полученное в теле ответа состояние корзины со своим текущим локальным состоянием пользовательской корзины и считает полученное состояние новым актуальным состоянием. После этого обработчик события завершает работу и возвращает управление в главный цикл событий (Eventloop).

Схема добавления товара в корзину представлена на рисунке 13.

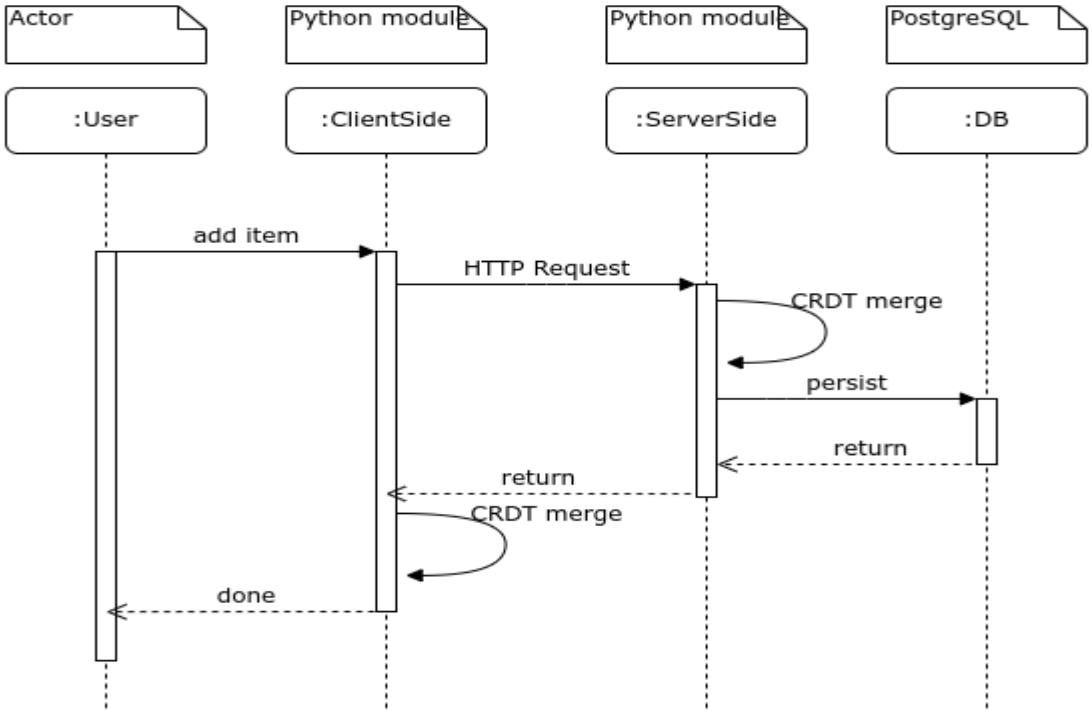

Рисунок 13 – Схема добавления товара в корзину товаров интернет магазина с использованием алгоритмов CRDT

Содержимое корзины хранится в памяти клиентского модуля, а при изменениях отправляется на сервер. В ответ сервер отправляет итоговую версию корзины, объединённую со своим внутренним состоянием в памяти. Для обеспечения резервирования корзина также сохраняется в реляционную базу данных.

Алгоритм работы клиентской части корзины товаров при добавлении/удалении товара представлен на рисунке 14.

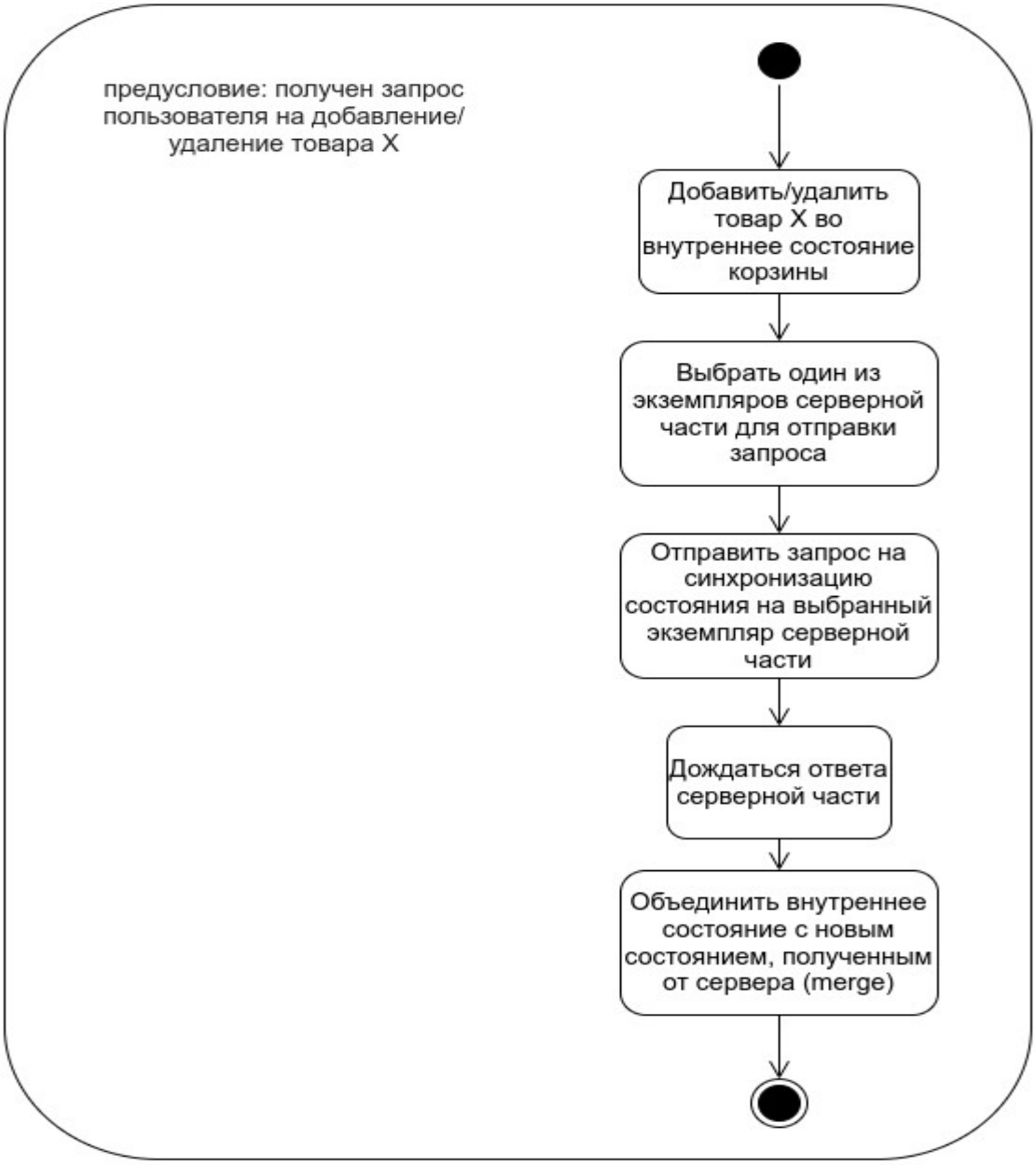

Рисунок 14 – Алгоритм работы клиентской части корзины товаров при удалении/добавлении элемента

Также клиентская часть корзины должна периодически синхронизировать состояние с сервером. Это позволяет учесть изменения,

внесённые на других клиентах. Алгоритм периодической синхронизации клиентской части представлен на рисунке 15.

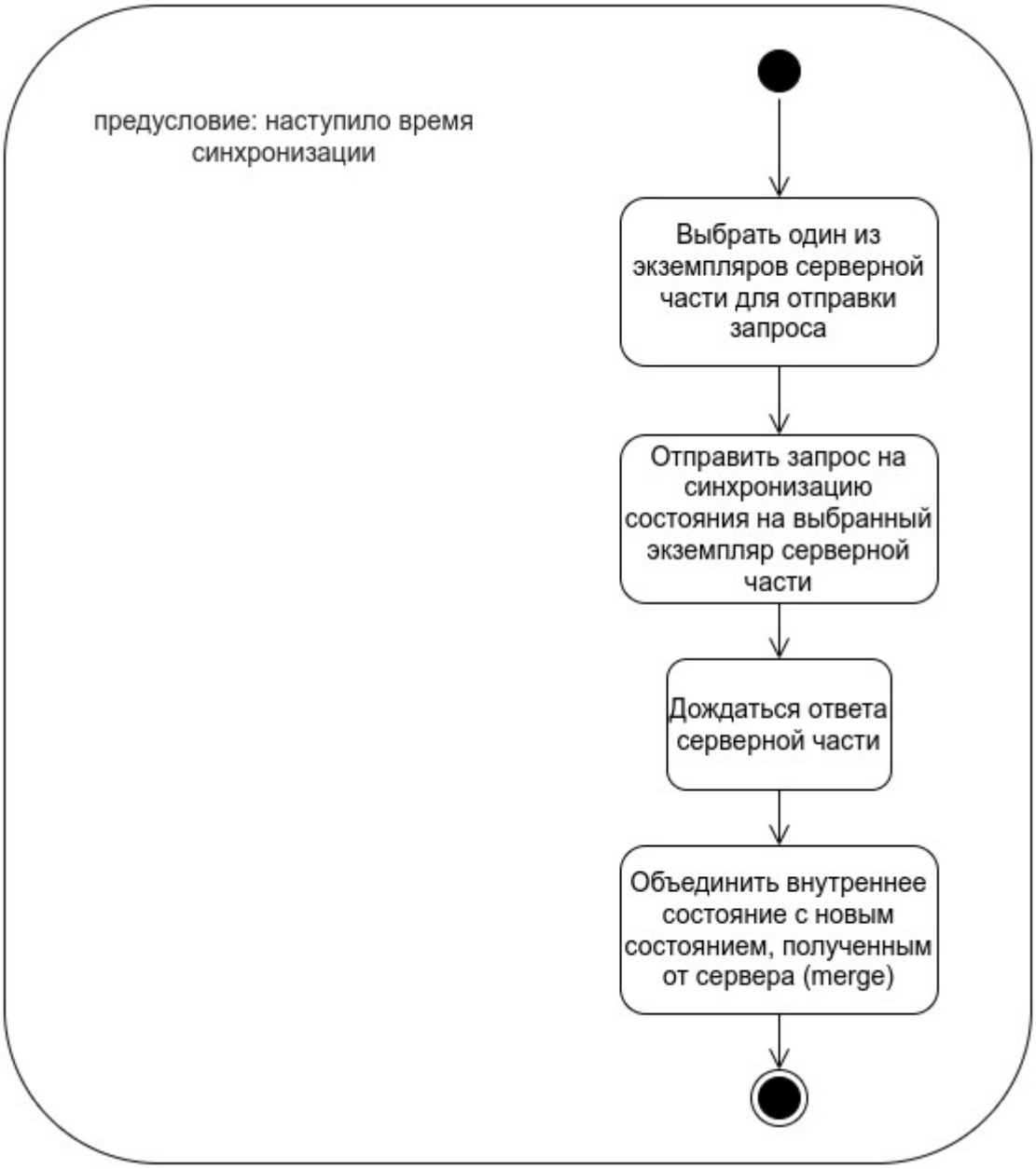

Рисунок 15 – Алгоритм периодической синхронизации клиентской части корзины

Так как клиентская часть хранит состояние в памяти, при получении списка товаров корзины возможно возвращать данные без совершения запроса к серверной части.

Алгоритм обработки запросов серверной части корзины представлен на рисунке 16.

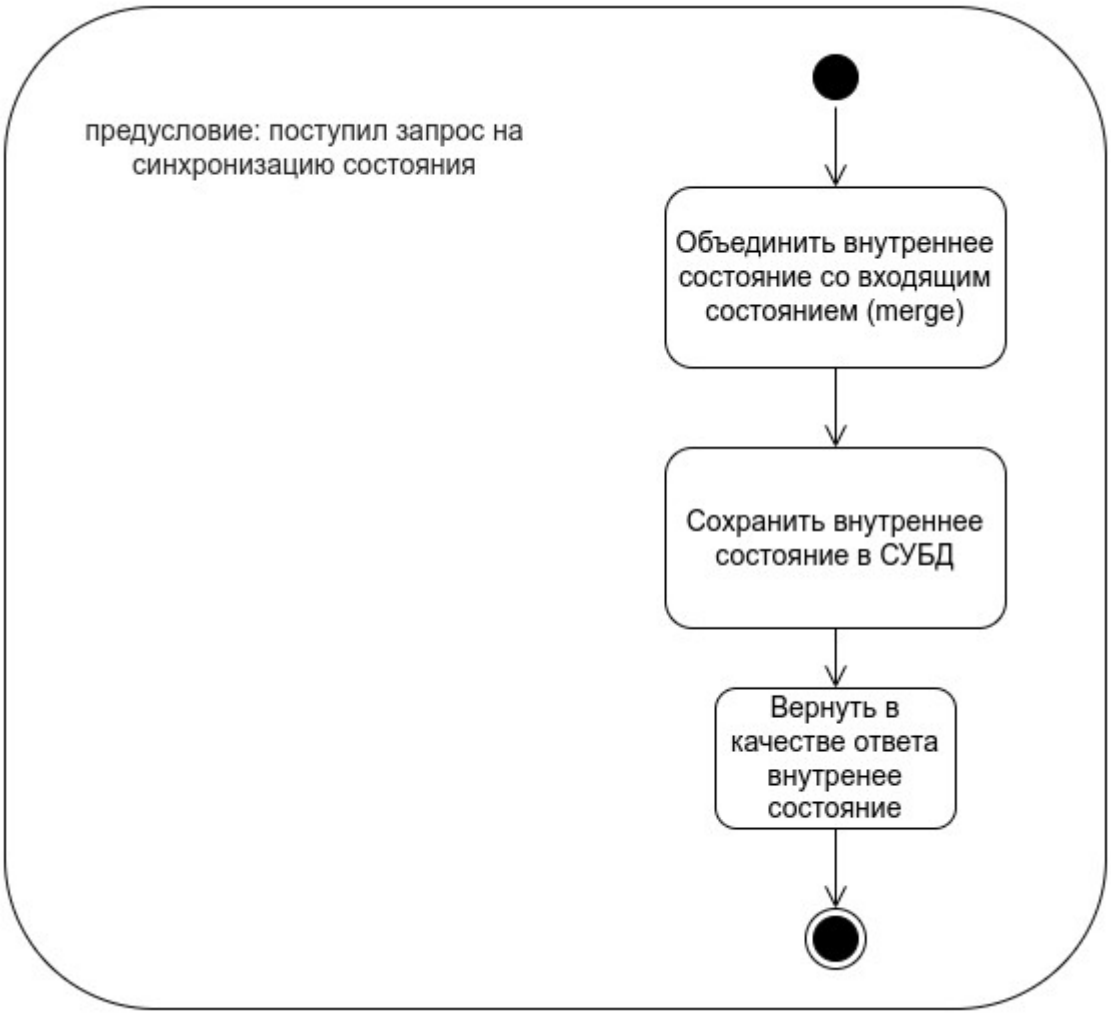

Рисунок 16 – Алгоритм работы серверной части корзины товаров интернет-магазина

Согласно представленным алгоритмам была разработана реализация корзины товаров интернет-магазина с использованием алгоритма CRDT OR Set. Диаграмма компонентов полученной реализации представлена на рисунке 17.

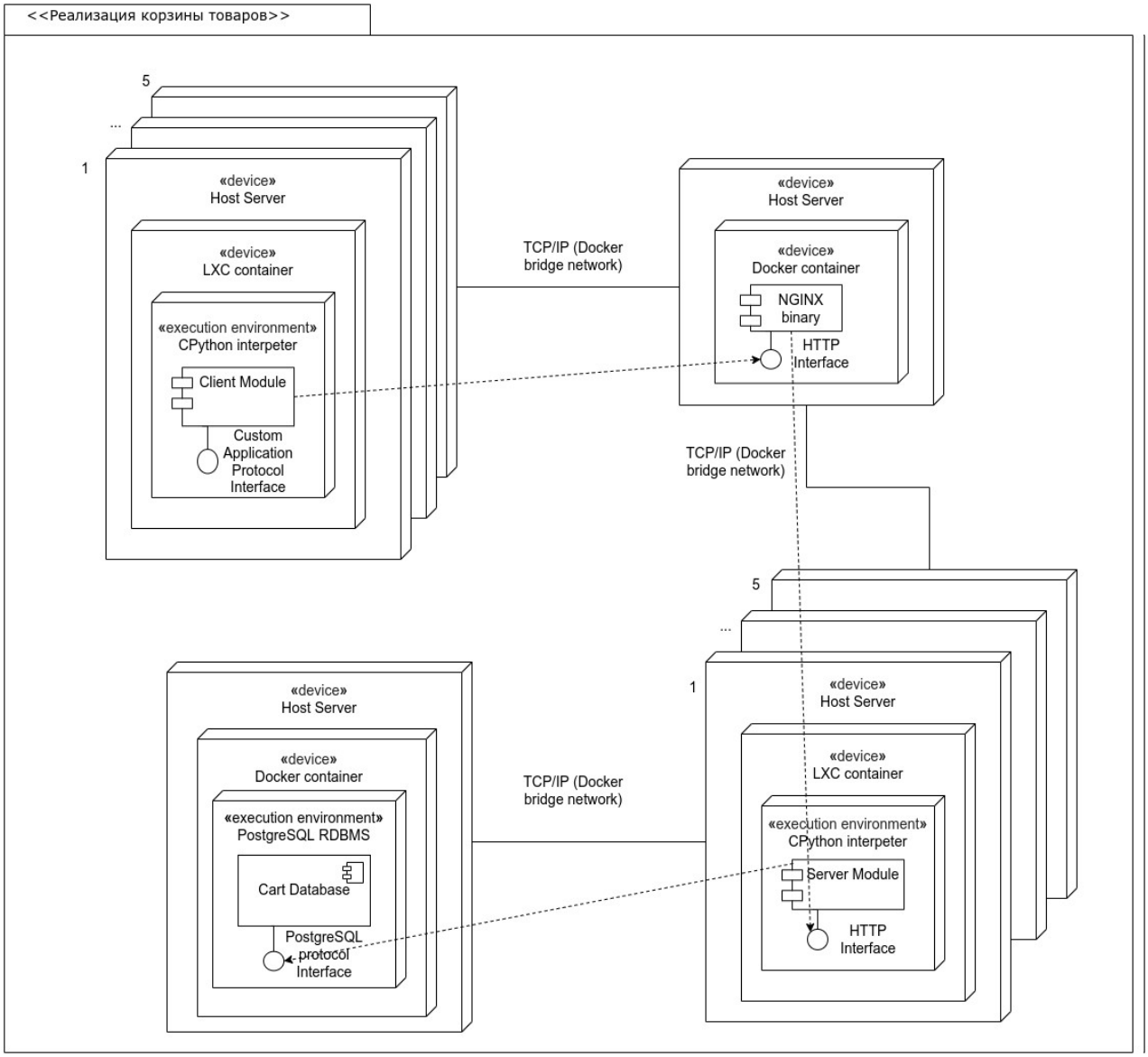

Рисунок 17 – Диаграмма компонентов реализации корзины товаров с использованием алгоритмов CRDT

Реализация состоит из клиентской и серверной части, а также общих модулей. Серверная часть представляет собой HTTP-сервер, использующий

базу данных PostgreSQL а также поддерживающий внутреннее состояние в оперативной памяти. Клиентская часть принимает команды по специально разработанному текстовому протоколу поверх TCP и отправляет соответствующие запросы на сервер, а также обновляет своё внутреннее состояние в памяти. Такой подход позволяет удобно автоматизировать взаимодействие с клиентом без использования средств браузерной автоматизации. В качестве формата передачи данных используется JSON. Используется виртуализация Docker и LXC. Исходный код доступен в репозитории по лицензии Eclipse Public License 2.0 [43].

Таким образом, была спроектирована и разработана реализация корзины товаров интернет-магазина с использованием алгоритмов CRDT. Алгоритмы работы полученной системы представлены на рисунках 14-16. Также разработана диаграмма компонентов (представлена на рисунке 17), включающая клиентскую и серверную части а также балансировщик нагрузки и реляционную СУБД для резервного копирования.

3.2 Проверка работоспособности полученной реализации в условиях сетевых разделений

Одно из главных преимуществ алгоритмов CRDT перед другими методами решения проблемы распределенной синхронизации данных в горизонтально масштабированных веб-приложениях — устойчивость к сетевым проблемам. Данное свойство позволяет использовать корзину товаров в условиях наличия множества мобильных клиентов с нестабльным соединениям. Для анализа практических преимуществ CRDT а также тестирования разработанной реализации использовался фреймворк Jepsen [21]. Jepsen — специализированный фреймворк для тестирования и верификации корректности распределенных систем с поддержкой fault injection (искусственно вызываемые сетевые проблемы) и многопоточности.

Фреймворк хорошо зарекомендовал себя, в частности с его помощью было найдено несколько concurrency-багов в таких популярных программных продуктах как Redis, PostgreSQL, MongoDB, Kafka, RabbitMQ, etcd, Zookeeper и многих других [22]. Фреймворк написан на языке программирования Clojure (современный диалект Lisp, исполняется на виртуальной машине Java (JVM)) и применяет функциональное программирование.

Для тестирования было развёрнуто 5 виртуальных узлов с использованием технологии LXC (Linux Containers) на дистрибутиве Linux Debian (релиз Buster). Характеристики хост-системы представлены в таблице 16.

Таблица 16 – Основные характеристики хост-системы для запуска виртуальных узлов

Характеристика	Значение
Дистрибутив хост-системы	Ubuntu 20.04.3 LTS
Процессор хост-системы	Intel i7-8650U @ 1.90GHz
BogoMIPS хост-системы	4199.88
Объем оперативной памяти хост-системы	16 Гб
Версия docker	20.10.7
Версия интерпретатора Python	3.8.10 (CPython)
Версия компилятора Go	1.14.4
Версия Java Runtime Environment	openjdk 14.0.2
Версия lxc	1:4.0.6
Дистрибутив LXC контейнеров	Debian 10.12.0
Docker-образ NGINX	nginx@sha256:bae781e7f518e0fb02245140c97e6ddc9f5fcf6aecc043dd9d17e33aec81c832
Docker-образ PostgreSQL	postgres:12
Версия Jepsen	0.2.1
Версия Clojure	1.10.0
Версия Leiningen	2.9.1

На каждом виртуальном узле были запущено по одному экземпляру клиентской и серверной части. Кроме того, для распределения нагрузки а

также большего соответствия реальным практическим развёртываниям клиентские части отправляли запросы на серверные части не напрямую, а через балансировщик нагрузки. В качестве балансировщика нагрузки использовался NGINX. В качестве базы данных использовался PostgreSQL развёрнутый в Docker-контейнере. Для запуска тестов использовался основной узел (ControlNode), с которого Jepsen взаимодействовал с другими узлами по протоколу SSH. Все скрипты для развёртывания и исходный код теста в виде Clojure-пакета доступны в репозитории по лицензии Eclipse Public License 2.0 [44]. Диаграмма компонентов стенда для тестирования приведена на рисунке 18. Основное отличие от базовой диаграммы, представленной на рисунке 17 — наличие Jepsen для проведения тестирования сетевых сбоев (выделено жёлтым цветом).

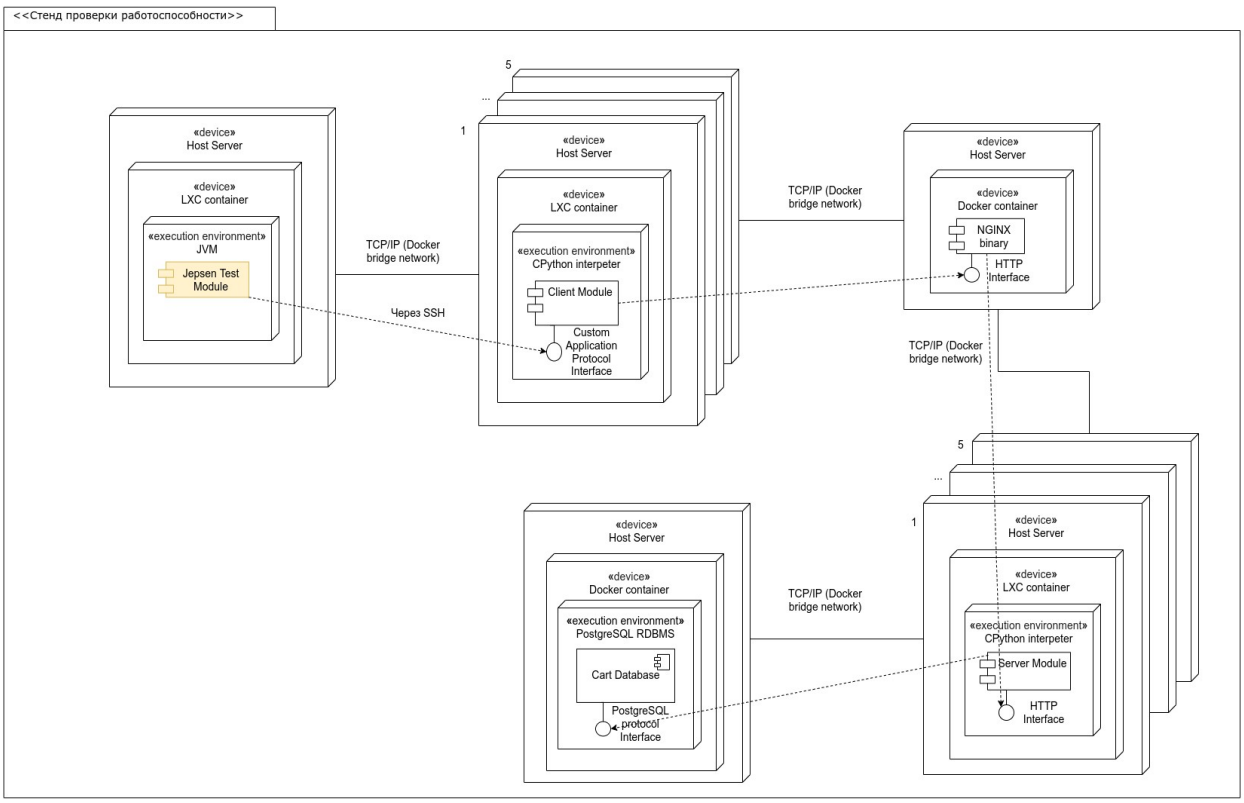

Рисунок 18 – Диаграмма компонентов стенда тестирования

Диаграмма добавления элемента в корзину в рамках среды тестирования представлена на рисунке 19.

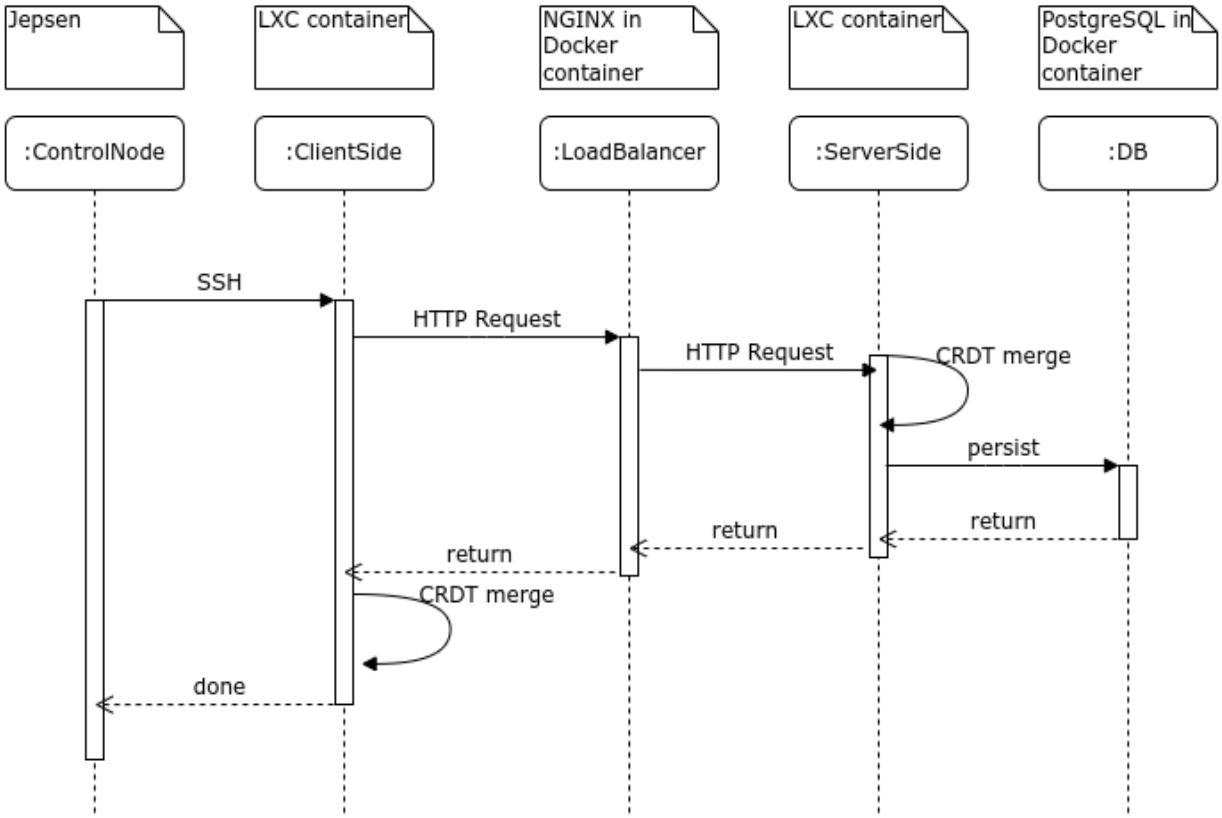

Рисунок 19 – Диаграмма добавления товара в корзину в рамках тестирования

Был реализован Jepsen-тест для тестирования и верификации полученной реализации. Тест запускается на 5 узлах. На каждом из узлов запускается случайный набор операций добавления и удаления товаров из корзины. Во время выполнения операций производится многократное случайное разделение узлов на сетевом уровне (т.е. узлы из группы 1 не могут связаться с узлами из группы 2 и наоборот) для имитации сетевых проблем и временно отключающихся клиентов. Затем производится восстановление сетевой связности и ожидание синхронизации всех узлов. После этого производится чтение результирующей корзины товаров.

Фактически считанное состояние корзины валидируется с использованием модели корзины. Для каждой операции обрабатывается ожидаемый результат операции вплоть до финального состояния. Затем результат обработки истории операций с точки зрения модели сравнивается с фактическим результатом чтения. Параметры теста могут указываться динамически при запуске.

Было проведено несколько запусков тестирования с постепенным возрастанием «сложности» (многопоточность, число узлов) и длительности. Основные результаты тестирования представлены в таблице 17. Под корректностью и успешностью теста понимается правильность итогового состояния системы:

- Все экземпляры корзины должны быть синхронизированы между собой;
- Каждый из добавленных товаров должен присутствовать в каждой корзине;
- Каждый из удалённых элементов должен отсутствовать в каждой корзине;
- Все запросы должны завершаться успехом.

Алгоритм имитации сетевых проблем работает следующим образом:

- Разделение всех узлов системы (т. е. в данном случае всех запущенных экземпляров) на две случайные группы;
- Внедрение на каждом узле сетевых правил, которые не позволяют группе 1 взаимодействовать с группой 2 и наоборот;
- Ожидание 5 секунд;

– Удаление сетевых правил и переход к шагу 1.

Таблица 17 – Результаты тестирования полученной реализации на корректность

Число виртуальных узлов	Число потоков на одном узле	Длительность теста, секунд	Внедренные сетевые проблемы	Тест успешен
5	1	120	Да	Да
5	10	120	Да	Да

На втором этапе тест запускался на пяти узлах и длился 120 секунд. Тест прошёл успешно.

Как известно, в практических развёртываниях веб-приложений один сервер обрабатывает запросы нескольких клиентов (конкретное число зависит от конфигурации и числа пользователей). Для того, чтобы получить максимально приближенный к реальным условиям тест использовалась многопоточность. На каждом узле запускалось 10 потоков тестирования, таким образом общее число потоков составляло 50. Длительность тестирования составляла 120 секунд. Тест прошёл успешно.

По результатам тестирования можно сделать следующие выводы:

- Удалось реализовать алгоритм CRDT OR Set с использованием современных технологий;
- При своей относительной простоте CRDT обладают высокой отказоустойчивостью: устойчивы к временно отключающимся клиентам а также сетевым проблемам которые могут возникать в реальных развёртываниях;

- CRDT хорошо выдерживают нагрузки, типичные для реальных веб-приложений (относительно небольшое число запросов для конкретной корзины одного пользователя, но большое общее число клиентских запросов для одного сервера).

Таким образом полученная реализация корзины товаров интернет-магазина с использованием алгоритмов CRDT была проверена на корректность. Для проверки использовался фреймворк тестирования распределенных систем Jepsen с функциями сетевой изоляции частей системы. По результатам проверки полученная реализация показывает корректное поведение даже при наличии сетевых проблем и разделении системы на изолированные части. Данное свойство является уникальным преимуществом алгоритмов CRDT перед альтернативными подходами к решению проблемы. Это свойство обосновывает возможность применения полученной реализации в условиях наличия множества мобильных клиентов с нестабильным соединением.

Глава 4 Анализ результатов решения проблемы эффективной синхронизации данных горизонтально масштабированных приложений с использованием алгоритмов CRDT

4.1 Развёртывание стенда для тестирования

Сравнительный анализ производительности производился на примере задачи синхронизации корзины пользователя интернет-магазина с использованием современных альтернативных программных продуктов (etcd), а также с учётом современных подходов к архитектуре веб-приложений. При проведении анализа использовались подходы синтетического тестирования в условиях, приближенных к реальным. Для повышения практической значимости исследования использовались современные технологии и актуальные подходы к разработке программного обеспечения. В качестве инструментов был выбран язык программирования Go [38].

Для сравнения используется базовая реализация с использованием алгоритмов CRDT (подход «CRDT»), а также три альтернативных подхода. Таким образом сравниваются следующие подходы к решению проблемы:

1. Подход «CRDT» (базовая реализация)
2. Подход «Классический» (использование одного мастер-сервера СУБД MySQL/MariaDB)
3. Подход «CRDT Master-Replica» (асинхронная репликация)
4. Подход «Кластерная нереляционная СУБД» (система с использованием распределенной системой хранения данных etcd)

Рассмотрим их подробнее.

4.1.1 Подход «CRDT»

Диаграмма компонентов системы представлена на рисунке 20. Диаграмма отличается от базовой диаграммы, представленной на рисунке 17 наличием автоматизированного клиента для тестирования (выделен жёлтым цветом).

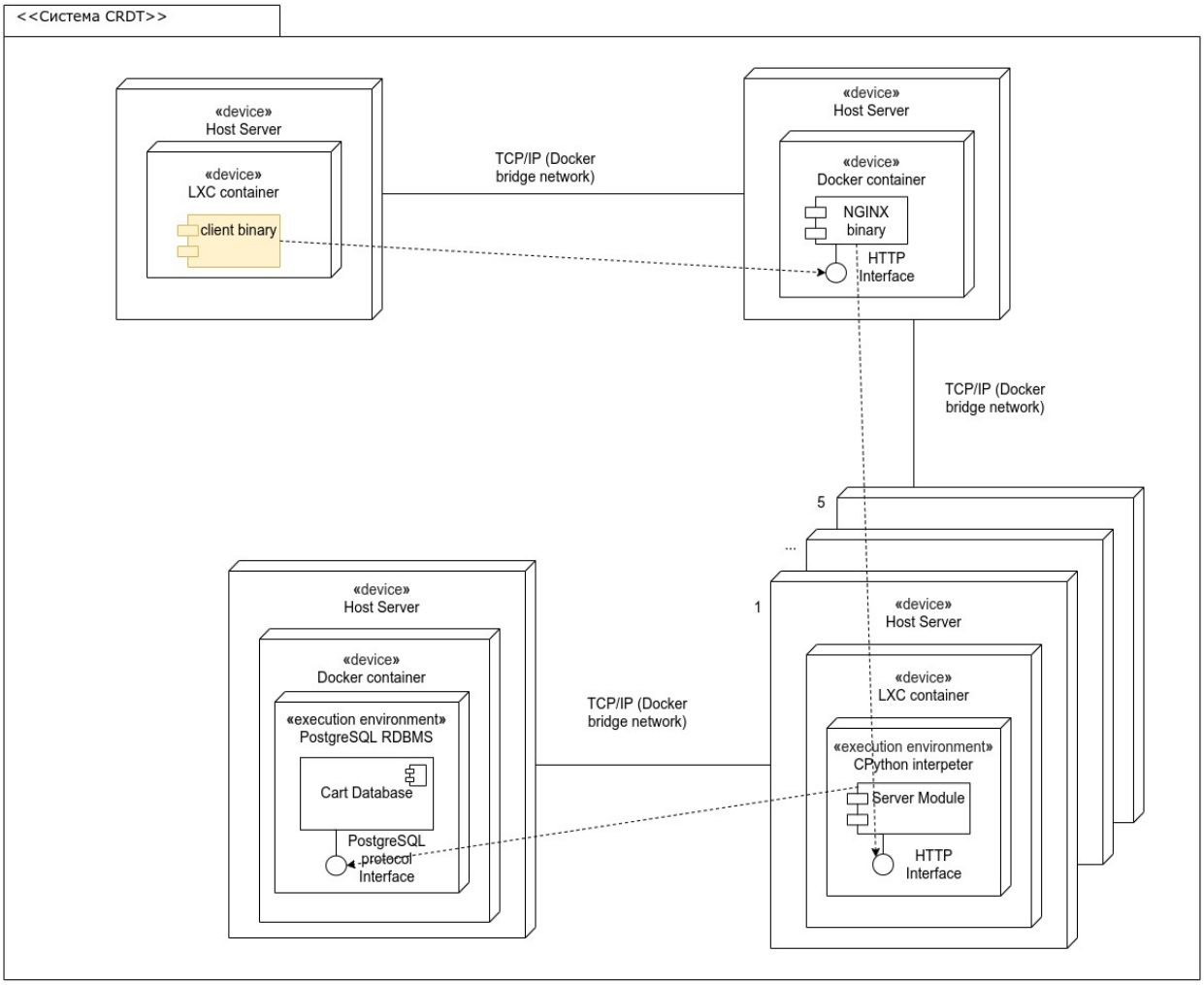

Рисунок 20 – Диаграмма компонентов системы «CRDT»

Алгоритм работы каждого из пяти экземпляров серверной части представлен на рисунке 21.

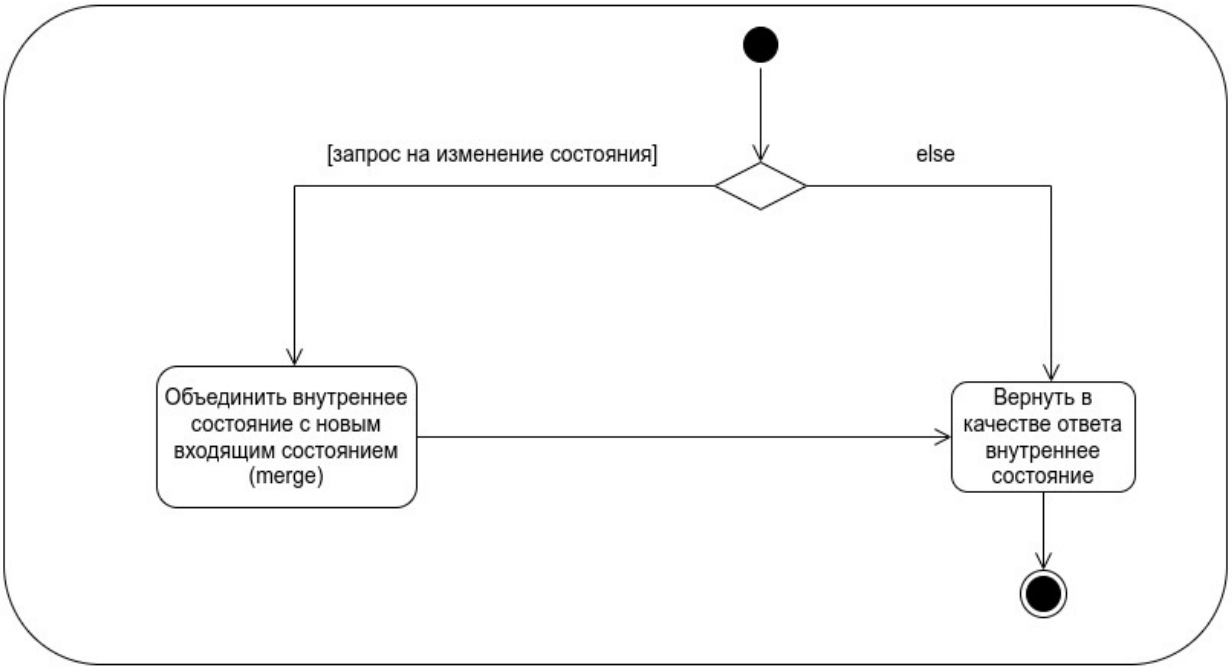

Рисунок 21 – Схема работы экземпляров серверной части системы «CRDT»

Данный подход использует полученную в предыдущем разделе CRDT-реализацию корзины товаров интернет-магазина. К реализации написан клиент на Go, который заполняет корзину в несколько потоков. В качества механизма управления зависимостями используется рекомендуемый механизм Go-модулей, минимальная версия языка для сборки клиента — 1.14.

4.1.2 Подход «Классический»

Данный подход использует классические технологии, применяемые в малых интернет-магазинах. А именно, используется один главный сервер популярной СУБД MariaDB (открытый форк MySQL) [42]. При подобной архитектуре обычно применяется асинхронная репликация — существует второй резервный сервер СУБД, который в фоновом режиме выполняет репликацию изменений и может стать основным сервером при прекращении

работы текущего основного сервера. Диаграмма компонентов системы представлена на рисунке 22.

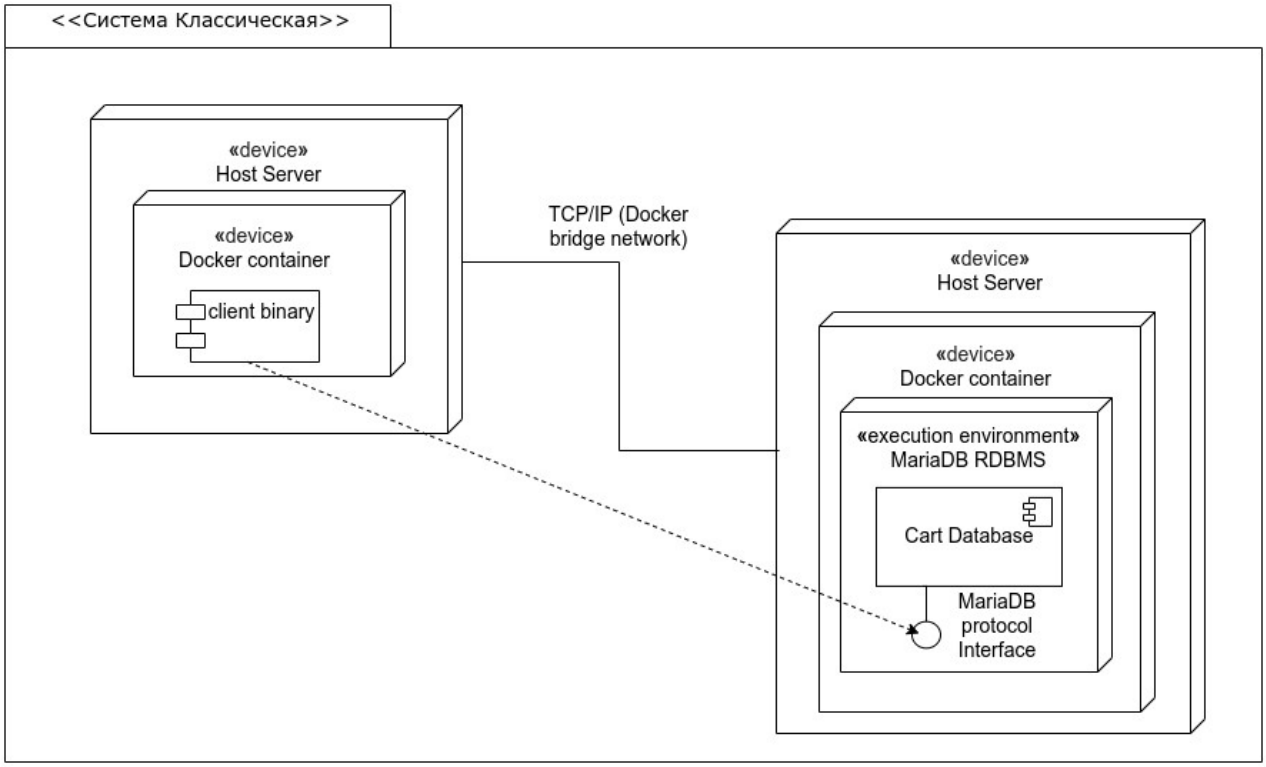

Рисунок 22 – Диаграмма компонентов системы «Классическая»

Для развёртывания использовался официальный Docker-образ mariadb:10.7.3.

4.1.3 Подход «CRDT Master-Replica»

Данный подход представляет собой модифицированную реализацию корзины товаров на основе алгоритма OR Set. В данной реализации определяется один главный экземпляр системы (мастер-узел). Мастер-узел используется для осуществления write-операций, то есть узлы-реплики проксируют запросы на изменение корзины на главный узел. При этом запросы на чтение корзины выполняются напрямую на репликах. Используется тот же Go-клиент.

Такая реализация соответствует асинхронной репликации (например асинхронной репликации в СУБД PostgreSQL), предполагается что мастер-узел в фоновом режиме «проталкивает» новые изменения корзины на реплики (push) или реплики в фоновом режиме «вытягивают» новые изменения корзины с мастер-узла (pull). В любом случае репликация осуществляется «в фоновом режиме» и не влияет на время обработки запросов — по этой причине конкретный механизм репликации реализован не был.

Диаграмма компонентов системы представлена на рисунке 23 (различие с предыдущей диаграммой выделено жёлтым цветом).

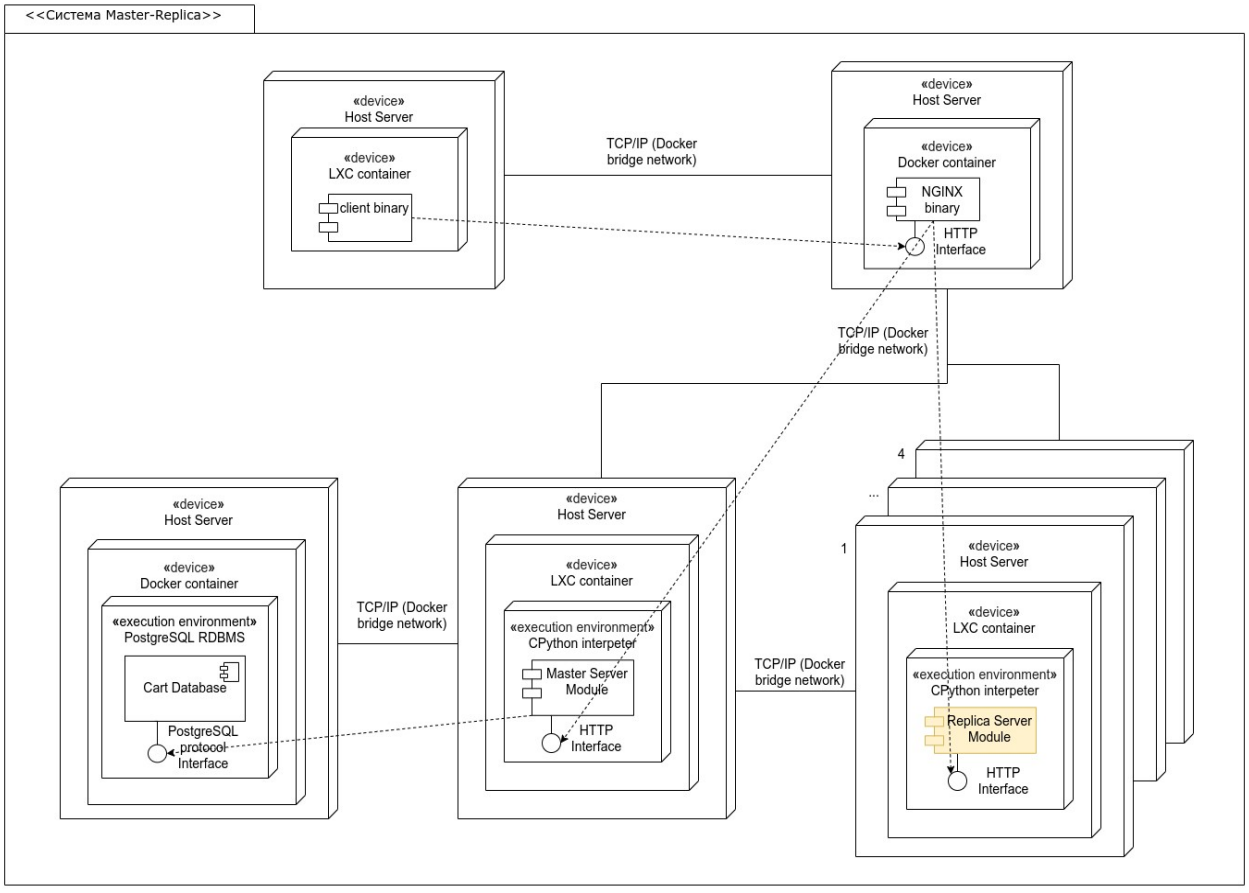

Рисунок 23 – Диаграмма компонентов системы «Master-Replica»

При этом мастер-узел системы работает по алгоритму, описанному на рисунке 21. Узлы-реплики работают по алгоритму, представленному на рисунке 24.

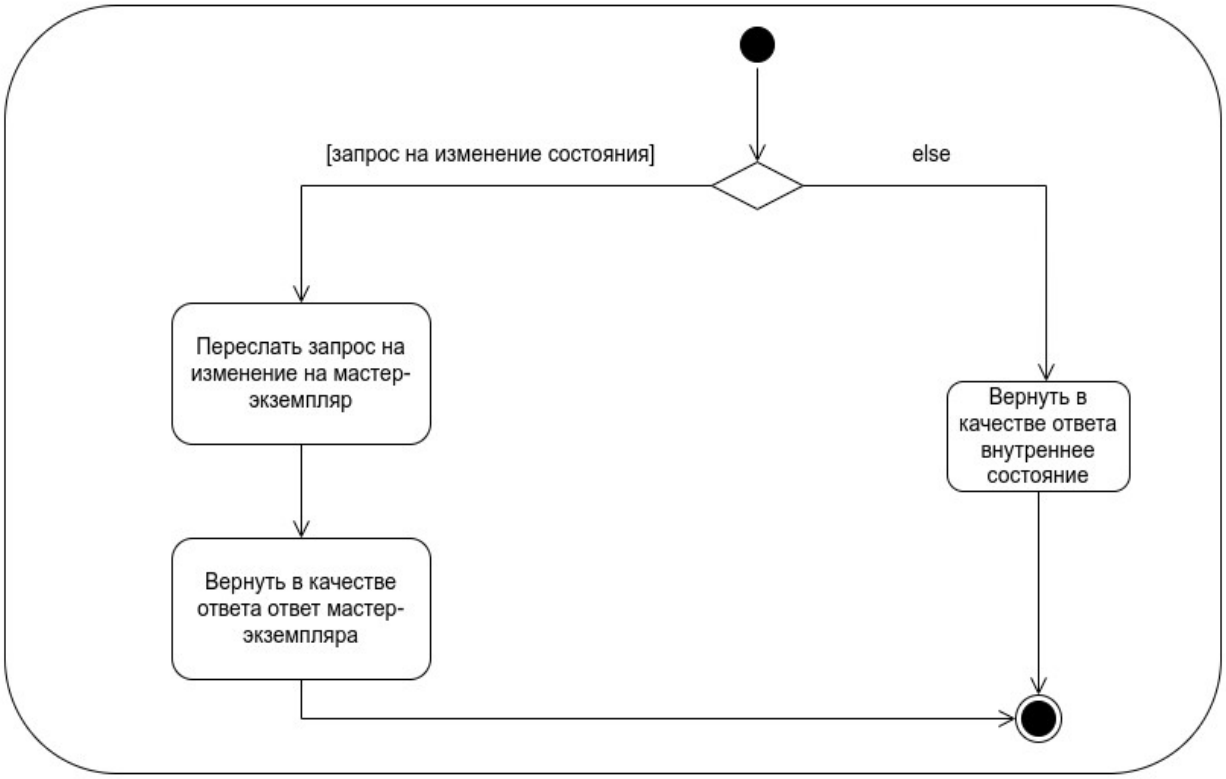

Рисунок 24 – Алгоритм работы вспомогательных экземпляров системы «Master-Replica»

Полученная реализация принципиально сопоставима с вариантом с внешними блокировками. В таком случае все реплики были бы равноправными, но для write-операций использовали бы внешний механизм синхронизации (например распределенные блокировки).

4.1.4 Подход «Кластерная нереляционная СУБД»

Данный подход использует популярную распределенную open-source систему управления конфигурацией etcd [13]. «Под капотом» etcd — хранилище пар «ключ-значение», а для обеспечения отказоустойчивости и согласованности данных используется алгоритм консенсуса Raft —

запущенные экземпляры etcd автоматически собираются в кластер и проводят выборы лидера (leader election). Etcd активно используется на практике — например именно эту систему использует для хранения данных популярный оркестратор контейнеров Kubernetes [29]. Система также предоставляет примитивы для блокировок и распределенных транзакций — они доступны через gRPC API а также через HTTP оболочку (gRPC gateway) [40]. Следует отметить что выбор лидера с помощью алгоритма консенсуса Raft использует критерий большинства. Поэтому для устойчивой работы etcd необходимо нечётное число узлов в кластере. В данном случае это условие выполняется.

Диаграмма компонентов системы представлена на рисунке 25. Использовался официальный Docker-образ quay.io/coreos/etcd:v3.5.1.

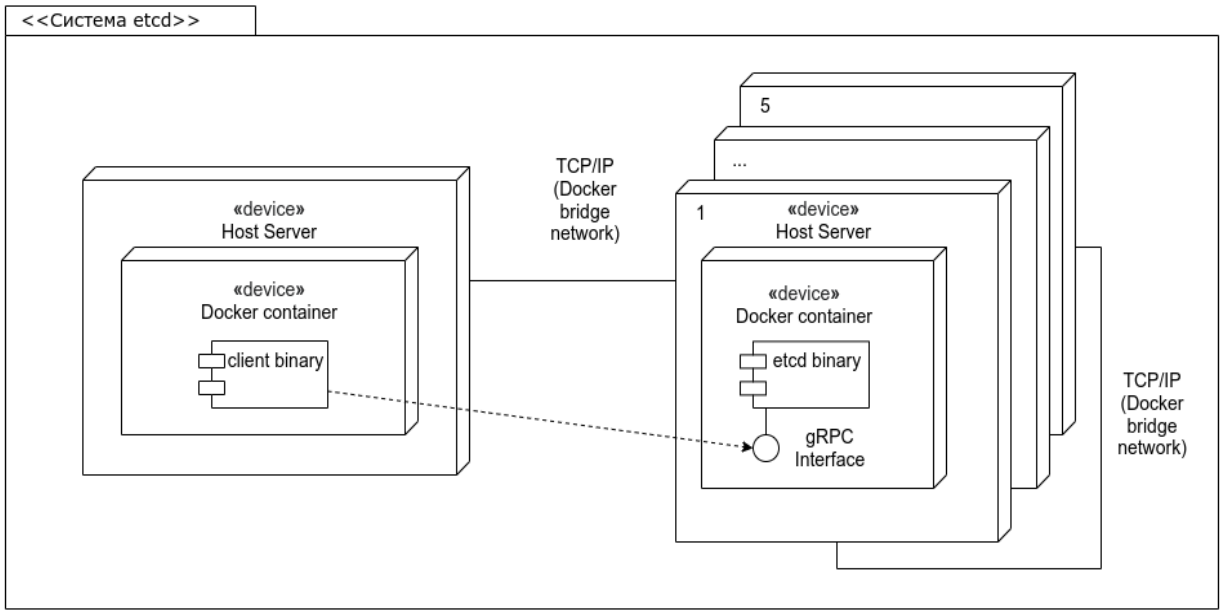

Рисунок 25 – Диаграмма компонентов системы «etcd»

Алгоритм работы каждого из экземпляров серверной части системы представлен на рисунке 26.

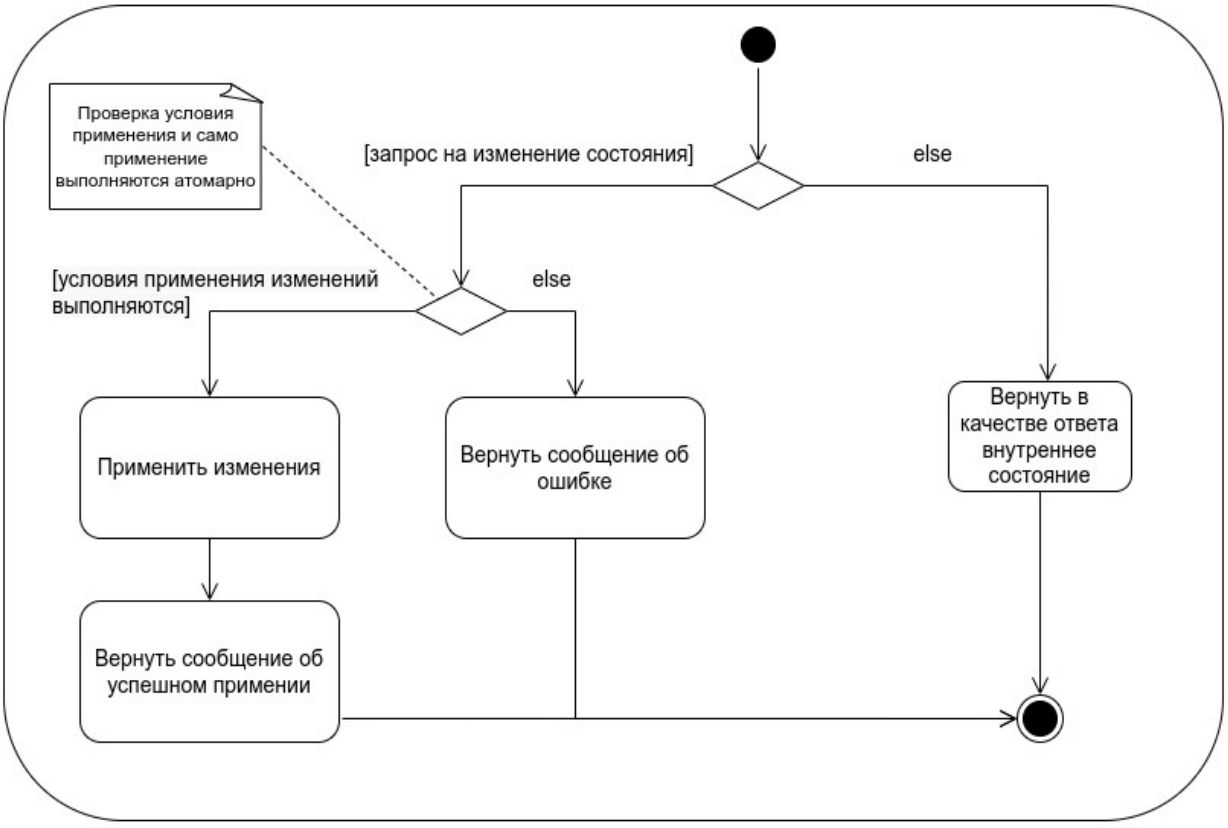

Рисунок 26 – Алгоритм работы экземпляров серверной части системы «etcd»

Используется официальный gRPC-клиент и функция Compare-And-Set (CAS) [12]. CAS позволяет добиться атомарных добавлений в корзину на всем кластере. Корзина хранится в значении ключа (в виде JSON-объекта), а каждый запрос атомарно сравнивает текущее состояние корзины с ожидаемым и добавляет элемент в корзину. Запрос успешно выполнится только если состояние корзины до добавления совпадает с ожидаемым, в противном случае добавление не будет произведено и запрос будет считаться неуспешным. То есть механизм Compare-And-Set позволяет реализовать распределенные транзакции на всем кластере etcd. Однако механизм CAS атомарен — если в условиях высокой нагрузки другой запрос успел изменить корзину, то текущий запрос не выполнится пока не подстроить ожидаемое состояние корзины под фактическое обновлённое состояние. Для этого

каждый запрос в подобных случаях может делать до 100 повторных попыток — перечитывать актуальное состояние корзины и пробовать добавить товар ещё раз уже в новое состояние.

Таким образом, для сравнения с альтернативными подходами к решению проблемы распределенной синхронизации корзины товаров интернет-магазина было выбрано 4 подхода:

- Базовый подход «CRDT» (рисунки 20-21);
- Подход «Классический» (рисунок 22);
- Подход «CRDT Master-Replica» (рисунки 23-24);
- Подход «Кластерная нереляционная СУБД» (рисунки 25-26).

Для избежания потери товаров в системе «etcd» был выбран механизм атомарных транзакций Compare-And-Set, описанный в [12].

4.2 Анализ производительности альтернативных вариантов решения проблемы эффективной синхронизации данных в горизонтально масштабированных веб-приложениях

Тестирование производится с использованием 8 потоков, для максимального использования современных многоядерных процессоров. При этом каждый поток асинхронный — Go использует кооперативную многозадачность, в частности горутина отдаёт управление другим горутинам на время ожидания ввода-вывода. Всего осуществляется 1000 добавлений товаров в корзину для подходов «CRDT» и «CRDT Master-Replica», 250 добавлений товаров в корзину для классического подхода и 100 добавлений товаров в корзину для подхода «Кластерная нереляционная СУБД». В каждом из подходов разворачивается 5 узлов (запущенных экземпляров серверной части веб-приложения): 5 равноправных узлов в «CRDT», 4 узла-реплики и 1 мастер-узел в «CRDT Master-Replica» и 5 динамически

координирующихся узлов в кластере etcd в системе «Кластерная нереляционная СУБД». В классическом подходе используется один запущенный экземлпяр MariaDB.

Для дополнительной проверки для подхода «CRDT» осуществляется проверка идентичности состояния всех узлов (на каждом из узлов есть все добавленные элементы). Перед этим осуществляется обмен содержимым корзин узлов для устранения влияния балансировки нагрузки. Время, затраченное на дополнительную синхронизацию и проверку не учитывается в общем времени. Для получения более точных результатов каждый тест запускается 10 раз, результаты усредняются.

Основные характеристики выбранных подходов представлены в таблице 18. Исходный код и инструкции по развёртыванию доступны в Github-репозитории [43].

Таблица 18 – Основные качественные и количественные характеристики рассматриваемых подходов

	Подход «CRDT»	Подход «Классический»	Подход «CRDT Master-Replica»	Подход «Кластерная нереляционная СУБД»
Реализация сервера	Однопоточный (ограничение GIL Python-интепретатора)	Многопоточный асинхронный (C++)	Однопоточный (ограничение GIL Python-интепретатора)	Многопоточный асинхронный (Go)
Реализация клиента	Многопоточный асинхронный (Go)	Многопоточный асинхронный (Go)	Многопоточный асинхронный (Go)	Многопоточный асинхронный (Go)

Продолжение таблицы 18

	Подход «CRDT»	Подход «Классический»	Подход «CRDT Master-Replica»	Подход «Кластерная нереляционная СУБД»
Топология узлов	5 полноправных узлов	1 мастер-узел	1 мастер-узел и 4 узла-реплики	1 мастер-узел и 4 узла-реплики (динамические роли)
Число потоков клиента (максимальное)	8	8	8	8
Число добавляемых товаров	1000	250	1000	100
Число запусков	10	10	10	10

Основные результаты приведены в таблице 2. Значения относительной длительности рассчитывались формуле 1.

$$\frac{\frac{1}{10} * \sum_{1}^{10} \left(t_{i(end)} - t_{i(start)} \right)}{\frac{1}{10} * \sum_{1}^{10} \left(t_{b(end)} - t_{b(start)} \right)} (1)$$

Где в числителе среднее время выполнения по 10 запускам для подхода X, в знаменателе — среднее время выполнения по 10 запускам для базового подхода (CRDT). Основные результаты приведены в таблице 19.

Таблица 19 – Основные результаты тестирования трёх распределенных систем

	Подход «CRDT»	Подход «Классический»	Подход «CRDT Master-Replica»	Подход «Кластерная нереляционная СУБД»
Средняя общая длительность выполнения запросов (секунд)	26.684	30.659	364.516	85.835
Во сколько раз медленнее подхода «CRDT»	1	1.149	13.66	3.217

Как видно из таблицы 19, подход «CRDT» на порядок быстрее обрабатывает заданное число запросов, чем подход «CRDT Master-Replica». Это подтверждает интуицию — нет «бутылочного горлышка» в виде единственного узла для всех пишущих запросов. Также подход «CRDT» более чем в 3 раза быстрее обрабатывает в 10 раз больший объем запросов, чем подход «Кластерная нереляционная СУБД». Такой эффект возникает из-за того, что при высокой нагрузке (множество одновременных запросов) распределенные блокировки которые возникают при операции Compare-And-Set оказываются не эффективными — запросы и их повторные попытки мешают друг другу. Аналогичный эффект возникает и в классическом подходе — подход «CRDT» обрабатывает в 4 раза больший объем запросов быстрее в 1.15 раз, чем классический подход так как при большом числе запросов из нескольких потоков запросы начинают мешать друг другу.

Следует отметить, что рассмотренные подходы обладают различным уровнем согласованности данных, данные различия приведены в таблице 20.

Таблица 20 – Различия в уровне согласованности данных среди рассматриваемых подходов

Название подхода	Уровень обеспечиваемой согласованности данных
Подход «CRDT»	Strong eventual consistency
Подход «Классический»	Strong consistency
Подход «CRDT Master-Replica»	Eventual consistency
Подход «Кластерная нереляционная СУБД»	Strong consistency

Как известно, алгоритмы CRDT обеспечивают eventual consistency. Подход «CRDT Master-Replica» как реализация асинхронной репликации обеспечивает такой же уровень консистентности, но подход «Классический» и подход «Кластерная нереляционная СУБД» обеспечивают более строгую консистентность. Этим могут объяснятся низкие результаты etcd в сценарии Compare-And-Set (более сильная консистентность требует больше блокировок и изоляции). По этой причине подходы не сравнимы напрямую, однако сравнение с etcd позволяет сопоставить полученную реализацию с лучшими образцами распределенных систем (пусть даже с более сильной консистентностью) и раскрыть потенциал.

Кроме различий в гарантиях согласованности данных, рассматриваемые подходы обладают различными программными реализациями. Данные различия могут оказать влияние на результаты сравнения. Различия приведены в таблице 21.

Таблица 21 – Различия в реализациях рассматриваемых подходов

Система	Язык программирования серверной части системы	Количество потоков серверной части системы	Поддержка асинхронного ввода-вывода	Число обрабатываемых запросов
Подход «CRDT»	Python	1	Нет	1000

Продолжение таблицы 21

Система	Язык программирования серверной части системы	Количество потоков серверной части системы	Поддержка асинхронного ввода-вывода	Число обрабатываемых запросов
Подход «Классический»	C++	>=1	Да	250
Подход «CRDT Master-Replica»	Python	1	Нет	1000
Подход «Кластерная нереляционная СУБД»	Go	>=1	Да	100

Сравниваемые подходы значительно отличаются в своей реализации. Серверная часть в рамках подхода «CRDT» и «CRDT Master-Replica» однопоточная и написана на интерпретируемом языке (Python), а реализации подходов «Классический» и «Кластерная нереляционная СУБД» поддерживают многопоточность и асинхронный ввод-вывод и исполняются в нативном коде. По этой причине в данном анализе мы избегаем прямого сопоставления с etcd и MariaDB, однако реализации «CRDT» и «CRDT Master-ReplicaCRDT Master-Replica» реализованы схоже, что позволяет сравнивать их напрямую. Также стоит отметить что различия в реализации играют на пользу подходам «Классический» и «Кластерная нереляционная СУБД», однако даже с таким преимуществом данные подходы оказывается медленнее — таким образом различия в реализации не искажают результаты анализа.

В рамках подхода «Кластерная нереляционная СУБД» обрабатывается 100 запросов, а не 1000 так как при значении 1000 запросы не могут успешно завершится за заданное число повторных попыток. Аналогично, в рамках подхода «Классический» обрабатывается 250 запросов, а не 1000 по этой же причине. Данную проблему можно обойти внедрением

рандомизированной задержки между попытками (например алгоритм exponential backoff с использованием джиттера), чтобы запросы меньше мешали друг другу и выросла вероятность успешно завершить CAS. Однако подобное изменение может также увеличить общее время обработки запросов. В данном случае меньшее число обрабатываемых запросов не влияет на результат так как даже с этим уменьшенным числом походы «Классический» и «Кластерная нереляционная СУБД» оказываются медленнее.

Также в состав программного окружения Go входит детектор состояний гонки (race-condition), использование которого позволяет снизить вероятность подобных ошибок (например, детектор обнаружил 42 состояния гонки в стандартной библиотеке языка [10]). Для дополнительной проверки кода клиента был осуществлён запуск с race-детектором. Состояний гонки обнаружено не было.

Таким образом был проведён анализ результатов с учётом особенностей реализации выбранных подходов. Подход с использованием алгоритмов CRDT обладает более высокой производительностью по сравнению с тремя другими рассматриваемыми подходами.

4.3 Ограничения проведённого тестирования

Проведённый анализ имеет свои ограничения и условия. Основные ограничения приведены в таблице 22.

Таблица 22 – Основные ограничения полученных результатов

Ограничение	Возможное влияние на результаты
Используемое при тестировании число потоков	При изменении числа потоков результаты могут изменится в любую сторону

Продолжение таблицы 22

Ограничение	Возможное влияние на результаты
Используемый при тестировании алгоритм балансировки нагрузки	При изменение алгоритма балансировки нагрузки для подхода «CRDT» результаты могут изменится в любую сторону
Неучтенные преимущества алгоритмов CRDT	Алгоритмы CRDT обладают рядом нефункциональных преимуществ, которые могут оказывать влияние на итоговое решение о внедрении алгоритмов CRDT на предприятии
Неучтенные программные оптимизации	Полученная реализация корзины интернет-магазина с использование алгоритмов CRDT может быть улучшена для достижения лучших результатов

Проведённый анализ имеет свои ограничения и условия. Проведённый анализ действителен для 8 потоков. Для другого количества потоков и/или другого количества процессорных ядер результаты могут быть другими.

Результаты действительны для метода балансировки round-robin и могут быть другими для других методов балансировки (так, возможно улучшение производительности подхода «CRDT» при использовании метода least-connections).

Неучтенные преимущества

Потенциально у алгоритмов CRDT могут быть и другие преимущества, трудно измеримые напрямую. Например, возможно ускорение скорости разработки за счет относительной простоты и математической обоснованности алгоритмов (по сравнению с алгоритмами Operational Transformation). Однако подобное ускорение (и даже его наличие) тяжело измерить на практике, тем более через синтетические тесты.

Другим потенциально неучтенным в данном анализе преимуществом может быть более низкая стоимость инфраструктуры. Подходы «Классический» и «CRDT Master-Replica» требуют вертикального

масштабирования, мастер-узел может быть только один, можно лишь наращивать его характеристики. Однако, как известно, вертикальное масштабирование имеет предел и чем ближе к пределу, тем дороже обходится наращивание мощности. В то же время подход «CRDT» позволяет горизонтальное масштабирование путём наращивания числа узлов. При этом несколько менее мощных серверов могут стоить дешевле, чем один сервер максимальной мощности. Тем не менее, конкретный экономический эффект сильно зависит от нагрузки и используемого аппаратного обеспечения.

Более низкая стоимость инфраструктуры может наблюдаться и в сравнении с etcd. Etcd активно использует диск и для эффективной работы требуются диски с низким показателем latency. Подобное требование менее важно для CRDT-реализации.

Нефункциональные преимущества полученной реализации приведены в таблице 23.

Таблица 23 – Нефункциональные преимущества полученной реализации корзины интернет-магазина на основе алгоритмов CRDT

Преимущество	Обоснование
Ускорение разработки	Относительная простота
Упрощение поддержки	Математическая обоснованность
Более низкая стоимость масштабирования	Проще осуществлять более дешёвое горизонтальное масштабирование
Более низкая стоимость используемых дисковых ресурсов	Возможность использовать менее производительные диски

Другие оптимизации

Стоит отметить, что полученная реализация корзины на основе алгоритмов CRDT может быть дополнительно оптимизирована. Возможные оптимизации представлены в таблице 24.

Таблица 24 – Возможные оптимизации полученной реализации корзины интернет-магазина на основе алгоритмов CRDT

Направление оптимизации	Сущность оптимизации
Повышение эффективности использования процессора серверной части	- Использование компилируемого в нативный код языка - Использование многопоточности
Повышение эффективности ввода-вывода	- Использование асинхронного ввода-вывода - Использование компактного формата для передачи данных

Подходы «CRDT» и «CRDT Master-Replica» оставляют широкий простор для оптимизаций. По этой причине абсолютные цифры не представляют собой максимальную производительность соответствующих систем. Тем не менее, относительные цифры сравнимы, так как в обеих системах используются сравнимые реализации. При внедрении подобных оптимизаций может вырасти относительное преимущество над системой «etcd», но результат сравнения не изменится.

Таким образом был проведён анализ ограничений проведённого тестирования, обоснована применимость результатов. Несмотря на значительные различия в реализации сравниваемых систем, эти различия не искажают результаты сравнения. В проведённом сравнении напрямую не учитывались нефункциональные преимущества, предоставляемые CRDT реализацией (простота разработки и поддержки, экономическая эффективность). Также не применялись технические оптимизации CRDT реализации. Перечисленные особенности могут дополнительно увеличить преимущество CRDT реализации.

Заключение

Даже с учётом ограничений алгоритмы CRDT обладают большим потенциалом для внедрения в некоторые подвиды веб-приложений, в особенности в веб-приложения интернет-магазинов для решения задачи синхронизации пользовательской корзины товаров горизонтально масштабированного веб-приложения.

Кроме того, при успешном внедрении алгоритмов CRDT можно ожидать улучшение производительности по сравнению с альтернативными вариантами решения проблемы распределенной синхронизации данных.

Основные выводы отражены в таблице 25.

Таблица 25 – Основные выводы

Вывод	Обоснование
В некоторых видах задач распределенной синхронизации данных возможно использовать практическую реализацию синхронизации через алгоритмы CRDT	Полученная реализация корзины интернет-магазина с использованием алгоритмов CRDT действительно решает задачу распределенной синхронизации данных, что подтверждается тестированием
Использование балансировщика нагрузки совместно с алгоритмами CRDT позволяет достичь автоматической синхронизации узлов в кластере	Полученная реализация орзины интернет-магазина с использованием алгоритмов CRDT не требует отдельного сервиса для обеспечения синхронизации узлов за счет использования балансировщика нагрузки
CRDT реализация корректна	Полученная реализация корзины интернет-магазина является корректной с точки зрения логического поведения корзины в соответствии с определёнными требованиями

Продолжение таблицы 25

Вывод	Обоснование
CRDT реализация устойчива к сетевым проблемам	Полученная реализация корзины интернет-магазина устойчива к сетевым проблемам связности между экземплярами горизонтально масштабированного приложения. Система выдерживает изоляцию экземпляров друг от друга без нарушений в работе.
CRDT реализация производительнее чем использование распределенных блокировок	Альтернативное решение проблемы с использованием одного ведущего узла для записи показывает меньшую производительность
CRDT реализация производительнее чем использование некоторых специализированных кластерых решений	Альтернативное решение проблемы с использованием специализированного кластерного хранилища ключ-значение etcd показывает меньшую производительность
CRDT реализация корзины обладает рядом нефункциональных преимуществ	По результатам анализа среди нефункциональных преимуществ CRDT реализации корзины перед альтернативными реализациями экономическая эффективность и простота поддержки
Алгоритмы CRDT обладают потенциалом для внедрения в некоторые виды современных веб-приложений	Алгоритмы CRDT могут использоваться для решения задачи распределенной синхронизации данных в нетребовательных к строгой согласованности веб-приложениях
Алгоритмы CRDT имеют особенно большой потенциал для решения задачи синхронизации корзины в веб-приложения интернет-магазинов	Специфические особенности бизнеса в сфере электронной коммерции предъявляют особые требования к стабильному хранению содержимого корзины

В рамках исследования была проанализирована проблема распределенной синхронизации данных в современных горизонтально масштабированных веб-приложениях. Существующие подходы к решению данной проблемы обладают серьёзными недостатками. Для решения проблемы и борьбы с недостатками существующих решений хорошо

подходит использование алгоритмов CRDT. Однако стоит учитывать, что при использовании алгоритмов CRDT в общем случае не обеспечивается строгая согласованность данных, как в классических реляционных СУБД. Поэтому алгоритмы CRDT применимы лишь в отдельном подмножестве веб-приложений, в которых нестрогая согласованность данных является приемлемой. Были разработаны подходы к определению таких веб-приложений, а также классификация веб-приложений в зависимости от потенциала применения алгоритмов CRDT.

По результатам анализа, одной из самых подходящих сфер для использования алгоритмов CRDT в качестве решения проблемы распределенной синхронизации горизонтально масштабированных веб-приложений является сфера электронной коммерции, а конкретно веб-приложения интернет-магазинов. При проработке применения алгоритмов CRDT в задаче синхронизации пользовательской корзины товаров удалось спроектировать корректную реализацию корзины с использованием алгоритмов CRDT. Был предложен способ достичь автоматической синхронизации узлов распределенного кластера с помощью использования балансировщика нагрузки. Полученная реализация показывает хорошие результаты с точки зрения производительности по сравнению с альтернативными подходами, а также обладает рядом нефункциональный преимуществ.

Список используемой литературы

[1] Active-Active Geo-Distribution (CRDTs-Based), [Электронный ресурс], URL: https://www.redislabs.com/redis-enterprise/technology/active-active-geo-distribution/ (дата обращения: 07.05.2022)

[2] Apps Are Becoming Distributed, What about Your Infra?, F5 Blog, 2020, [Электронный ресурс], URL: https://www.f5.com/company/blog/apps-are-becoming-distributed-what-about-your_infra (дата обращения: 07.05.2022)

[3] Automerge: A JSON-like data structure (a CRDT) that can be modified concurrently by different users, and merged again automatically., [Электронный ресурс], URL: https://github.com/automerge/automerge (дата обращения: 07.05.2022)

[4] Bieniusa A. и другие, An optimized conflict-free replicated set [Research Report] RR-8083, Inria – Centre Paris-Rocquencourt; INRIA. 2012, стр. 3.

[5] Cabrita G. M. Non-uniform replication for replicated objects : дис. – 2017.

[6] C. Ellis and S. J. Gibbs, Concurrency control in groupware systems, //ACM International Conference on Management of Data (SIGMOD), May 1989. – С. 399–407.

[7] Dharma Shukla, Azure Cosmos DB: Pushing the frontier of globally distributed databases, [Электронный ресурс], URL: https://azure.microsoft.com/en-us/blog/azure-cosmos-db-pushing-the-frontier-of-globally-distributed-databases/ (дата обращения: 07.05.2022)

[8] Digital Adoption Index, World Bank, [Электронный ресурс], URL: https://www.worldbank.org/en/publication/wdr2016/Digital-Adoption-Index (дата обращения: 07.05.2022)

[9] Dmitry Ivanov, Practical Data Synchronization with CRDTs, StrangeLoop Conference 2016, [Электронный ресурс], URL: https://speakerdeck.com/ajantis/practical-data-synchronization-with-crdts-strangeloop-2016 (дата обращения: 07.05.2022)

[10] Dmitry Vyukov, Andrew Gerrand, Introducing the Go Race Detector, [Электронный ресурс], URL: https://go.dev/blog/race-detector (дата обращения: 07.05.2022)

[11] Doug Woos и другие, Planning for Change in a Formal Verification of the Raft Consensus Protocol, Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, стр. 154-165

[12] etcd3 API - Transaction, Официальная документация etcd, [Электронный ресурс], URL: https://github.com/etcd-io/website/blob/v3.5.0/content/en/docs/v3.5/learning/api.md#transaction (дата обращения: 07.05.2022)

[13] etcd. Distributed reliable key-value store for the most critical data of a distributed system. [Электронный ресурс]. URL: https://github.com/etcd-io/etcd (дата обращения: 07.05.2022)

[14] Evan Wallace, How Figma’s multiplayer technology works, [Электронный ресурс], URL: https://www.figma.com/blog/how-figmas-multiplayer-technology-works/ (дата обращения: 07.05.2022)

[15] Faiz M., Shanker U. Data synchronization in distributed client-server applications //2016 IEEE International Conference on Engineering and Technology (ICETECH). – IEEE, 2016. – С. 611-616.

[16] Galera Cluster for MySQL, [Электронный ресурс], URL: https://galeracluster.com/ (дата обращения: 07.05.2022)

[17] Gene T. J. Wuu и Arthur J. Bernstein. Efficient solutions to the replicated log and dictionaryproblems. InSymp. on Principles of Dist. Comp. (PODC), стр. 233–242, Vancouver, BC, Canada, 1984.

[18] Giuseppe DeCandia и другие, Dynamo: amazon's highly available key-value store, ACM SIGOPS Operating Systems Review, Vol. 41, No. 6 (2007), стр. 205-220

[19] G. Oster, P. Urso, P. Molli, and A. Imine, Data consistency for P2P collaborative editing, //ACM Conference on Computer Supported Cooperative Work (CSCW), Banff, Alberta, Canada, ACM Press, 2006. – С. 399–407.

[20] He W., Da Xu L. Integration of distributed enterprise applications: A survey //IEEE Transactions on industrial informatics. – 2012. – Т. 10. – №. 1. – С. 35-42.

[21] Jepsen. A framework for distributed systems verification, with fault injection, [Электронный ресурс], URL: https://github.com/jepsen-io/jepsen (дата обращения: 07.05.2022)

[22] Jepsen. Analyses, [Электронный ресурс], URL: http://jepsen.io/analyses (дата обращения: 07.05.2022)

[23] Marc Shapiro и другие, Convergent and Communicative Replicated Data Types, Bulletin – European Association for Theoretical Computer Science (2011), стр. 67-88

[24] Michal Ptaszek, Scaling LoL Chat to 70M Players, StrangeLoop Conference 2014, [Электронный ресурс], URL: https://www.slideshare.net/michalptaszek/strange-loop-presentation (дата обращения: 07.05.2022)

[25] MySQL 8.0 Reference Manual: Chapter 17 Replication, Официальная документация MySQL, [Электронный ресурс], URL: https://dev.mysql.com/doc/refman/8.0/en/replication.html

[26] MySQL 8.0 Reference Manual: XA Transactions, Официальная документация MySQL, [Электронный ресурс], URL: https://dev.mysql.com/doc/refman/8.0/en/xa.html

[27] Nuno Preguica и другие, Conflict-free Replicated Data Types, Symposium on Self-Stabilizing Systems 2011: Stabilization, Safety, and Security of Distributed Systems (2011), стр. 386-400

[28] Online editor providing collaborative editing in really real-time using CRDTs and IPFS., [Электронный ресурс], URL: https://github.com/peer-base/peer-pad (дата обращения: 07.05.2022)

[29] Operating etcd clusters for Kubernetes, Официальная документация Kubernetes, [Электронный ресурс], URL: https://kubernetes.io/docs/tasks/administer-cluster/configure-upgrade-etcd/ (дата обращения: 07.05.2022)

[30] Peter Bourgon, Roshi: a CRDT system for timestamped events, [Электронный ресурс], URL: https://developers.soundcloud.com/blog/roshi-a-crdt-system-for-timestamped-events (дата обращения: 07.05.2022)

[31] Pew Research Center, Spring 2018 Global Attitudes Survey Q46 [Электронный ресурс] URL: https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/ (дата обращения 07.05.2022)

[32] PostgreSQL 14 Documentation: 27.2. Log-Shipping Standby Servers, Официальная документация PostgreSQL, [Электронный ресурс], URL: https://www.postgresql.org/docs/current/warm-standby.html

[33] P. R. Johnson and R. H. Thomas. The maintenance of duplicate databases. Internet Request for Comments RFC 677, Information Sciences Institute, Jan. 1976.

[34] Rana Asif Rehman, Bilal Khan и др, IoT Elements, Layered Architectures and Security Issues: A Comprehensive Survey, Sensors 18(9), 2018, стр. 1-37

[35] Shapiro M. И другие. Conflict-free replicated data types //Symposium on Self-Stabilizing Systems. – Springer, Berlin, Heidelberg, 2011. – С. 386-400.

[36] Statista, Mobile Internet & Apps, [Электронный ресурс], URL: https://www.statista.com/markets/424/topic/538/mobile-internet-apps/#overview (дата обращения: 07.05.2022)

[37] String-wise sequence CRDT powering peer-to-peer collaborative editing in Teletype for Atom., [Электронный ресурс], URL: https://github.com/atom/teletype-crdt (дата обращения: 07.05.2022)

[38] The Go Programming Language, Официальный сайт Go, [Электронный ресурс], URL: https://go.dev/ (дата обращения: 07.05.2022)

[39] The Open Group, Distributed Transaction Processing: The XA Specification, официальный сайт публикаций The Open Group, [Электронный ресурс], URL: https://pubs.opengroup.org/onlinepubs/009680699/toc.pdf

[40] Why gRPC gateway, Официальная документация etcd, [Электронный ресурс], URL: https://github.com/etcd-io/website/blob/v3.5.0/content/en/docs/v3.5/dev-guide/api_grpc_gateway.md (дата обращения: 07.05.2022)

[41] Yjs: Peer-to-peer shared types, [Электронный ресурс], URL: https://github.com/yjs/yjs (дата обращения: 07.05.2022)

[42] Документация MariaDB, (официальный сайт MariaDB Foundation), [Электронный ресурс], URL: https://mariadb.org/documentation/

[43] Репозиторий исходного кода. [Электронный ресурс], URL: https://github.com/nikitakalyanov/crdt-cart (дата обращения: 07.05.2022)

[44] Репозиторий исходного кода Jepsen-теста, [Электронный ресурс], URL: https://github.com/nikitakalyanov/jepsen.crdtcart (дата обращения: 07.05.2022)