ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

На правах рукописи

Азиз Аммар Имад Азиз

СПЕЦИАЛЬНОЕ МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПРОЦЕССА БЕЗОПАСНОГО УПРАВЛЕНИЯ РЕПЛИКАЦИЯМИ В МАСШТАБИРУЕМЫХ СУБД

Специальность: 2.3.5. Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель:
д.т.н., доцент Бондаренко Юлия Валентиновна

Воронеж - 2023

Оглавление

Введение

Актуальность темы. Объем данных, генерируемых мобильными гаджетами, датчиками и другими вычислительными устройствами, явился причиной создания концепции больших данных.

В этом контексте СУБД NoSQL стали популярными благодаря большей производительности для управления большими наборами данных с помощью альтернативных моделей данных. Многие службы облачных вычислений используют СУБД NoSQL, но оценка качества обслуживания (QoS) создает дополнительные проблемы для этих систем. Такими исследованиями занимались Вишневский В.М., Денисов А.А., Кравец О.Я., Мелентьев В.А., Bruneo D., Chakraborty S., Mohan, C. Интересным является подход, основанный на обобщенных стохастических сетях Петри (GSPN) для обеспечения производительности частных облачных вычислительных сред, которые принимают СУБД NoSQL как систему хранения.

Развитие Интернет-вещей (IoT) и популяризация мобильных устройств значительно поспособствовали созданию новой концепции, а именно Больших данных, которая характеризует скорость, объем и разнообразие данных. В этом контексте СУБД NoSQL стали замечательной альтернативой реляционным СУБД для решения проблем с большими данными. СУБД NoSQL обычно обладают лучшей производительностью для обработки больших наборов данных, их легко развернуть как распределенную систему, и они более эффективно управляют неструктурированными данными. Кроме того, NoSQL позволяет устанавливать различные уровни согласованности между резервными серверами.

Следует отметить, что многие сервисы облачных вычислений используют NoSQL, но существуют дополнительные проблемы для оценки

показателей качества обслуживания (QoS), например, из-за различных рабочих нагрузок и влияния доступности. При этом интересной представляется идея совместной оценки эффективности и доступности (т.е. производительности) частных облачных вычислительных систем, использующих СУБД NoSQL в качестве подсистемы хранения.

В этой связи, актуальность исследования определяется необходимостью разработки дополнительных компонентов математического и программного обеспечения процесса безопасного управления репликациями в масштабируемых СУБД на основе реализации соответствующих алгоритмических средств.

Тематика диссертационной работы соответствует научному направлению ФГБОУ ВО «Воронежский государственный университет» «Суперкомпьютерные технологии, квантовые и распределенные вычисления, большие данные».

Целью работы является развитие средств математического и программного обеспечения процесса безопасного управления репликациями в масштабируемых СУБД на основе разработки специального мультиагентного самонастраивающегося алгоритма.

Задачи исследования. Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать модель частных облачных вычислительных систем с NoSQL-подсистемами хранения данных на основе обобщенных стохастических сетей Петри.

2. Создать алгоритм динамической балансировки системы, основанный на многокритериальном принятии решений при формировании оптимальной матрицы альтернатив.

3. Разработать мультиагентный самонастраивающийся алгоритм репликации, основанный на обучении с подкреплением с использованием монитора метрик, диспетчера обновлений и агентов обучения с

подкреплением.

4. Создать сервис-ориентированную архитектуру несвязанной СУБД с модифицированной подсистемой ведения журнала транзакций и механизмом его очистки с нулевой памятью.

5. Разработать архитектуру человеко-машинной системы проектирования базовой структуры БД с интерактивным характером выбора оптимальных характеристик межсистемного взаимодействия.

Объект исследования: системы облачных вычислений.

Предмет исследования: математические средства улучшения безопасного управления репликациями в масштабируемых СУБД.

Методы исследования. При решении поставленных в диссертации задач использовались теория вероятностей, теория принятия решений, а также методы объектно-ориентированного программирования.

Тематика работы соответствует следующим пунктам паспорта специальности 2.3.5 «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»: п.3 «Модели, методы, архитектуры, алгоритмы, языки и программные инструменты организации взаимодействия программ и программных систем»; п.9 «Модели, методы, алгоритмы, облачные технологии и программная инфраструктура организации глобально распределенной обработки данных».

Научная новизна работы. В диссертации получены следующие результаты, характеризующиеся научной новизной:

1. Модель частных облачных вычислительных систем с подсистемами хранения данных на основе NoSQL, отличающаяся использованием обобщенных стохастических сетей Петри и обеспечивающая корректную оценку пропускной способности системы при различных рабочих нагрузках, доступности и простоях

2. Алгоритм динамической балансировки системы, основанный на

многокритериальном принятии решений при получении оптимальной матрицы альтернатив и обеспечивающий повышение производительности и QoS с одновременным определением приоритетов процессов.

3. Мультиагентный самонастраивающийся алгоритм репликации, основанный на обучении с подкреплением с использованием монитора метрик, диспетчера обновлений и агентов обучения с подкреплением, обеспечивающий конфигурирование гибридного промежуточного программного обеспечения репликации без перезапуска СУБД в режиме реального времени.

4. Сервис-ориентированная архитектура несвязанной СУБД, отличающаяся модифицированной подсистемой ведения журнала транзакций и механизмом его очистки с нулевой памятью, обеспечивающая снижение служебного сетевого трафика и повышение безопасности обработки при проектировании разделения вычислений и данных.

5. Архитектура человеко-машинной системы проектирования базовой структуры БД, отличающаяся интерактивным характером выбора оптимальных характеристик межсистемного взаимодействия и обеспечивающая автоматизацию динамического включения новых подмоделей в структуру БД.

Теоретическая и практическая значимость исследования в разработке средств специального математического и программного обеспечения процесса безопасного управления репликациями в масштабируемых СУБД на основе многокритериального принятия решений при получении оптимальной матрицы альтернатив, а также средств информационного и программного обеспечения для экспериментальной оценки качества разработанных алгоритмов.

Теоретические результаты работы могут быть использованы в проектных и научно-исследовательских организациях, занимающихся

проектированием платформенно-инвариантных систем управления облачными средами в условиях штатной или нерегламентированной внешней нагрузки.

Положения, выносимые на защиту

1. Модель частных облачных вычислительных систем с подсистемами хранения данных на основе NoSQL обеспечивает корректную оценку пропускной способности системы при различных рабочих нагрузках, доступности и простоях

2. Алгоритм динамической балансировки системы обеспечивает повышение производительности и QoS с одновременным определением приоритетов процессов.

3. Мультиагентный самонастраивающийся алгоритм репликации, основанный на обучении с подкреплением, позволяет осуществить конфигурирование гибридного промежуточного программного обеспечения репликации без перезапуска СУБД в режиме реального времени.

4. Сервис-ориентированная архитектура несвязанной СУБД с модифицированной подсистемой ведения журнала транзакций обеспечивает снижение служебного сетевого трафика и повышение безопасности обработки при проектировании разделения вычислений и данных.

5. Архитектура человеко-машинной системы проектирования базовой структуры БД обеспечивает автоматизацию динамического включения новых подмоделей в структуру БД.

Результаты внедрения. Основные результаты внедрены в виде программных компонент систем управления транзакциями в АО НПП «РЕЛЭКС» (г. Воронеж), «Al Rawiyah for information technology» company (Багдад, Ирак), а также в учебный процесс Воронежского государственного университета в рамках дисциплин: «Базы данных»,

«Информационная безопасность и защита информации», «Информатизация предприятия», а также в рамках курсового и дипломного проектирования.

Апробация работы. Основные положения диссертационной работы докладывались и обсуждались на следующих конференциях: Всероссийской научной конференции «Достижения науки и технологий-ДНиТ-2021» (Красноярск, 2021); Международной научной конференции «Актуальные проблемы прикладной математики, информатики и механики» (Воронеж, 2021); XXVII-th International Open Science Conference «Modern informatization problems in economics and safety (MIP-2022'ES)» (Yelm, WA, USA, 2022), Международной научной конференции «Актуальные проблемы прикладной математики, информатики и механики» (Воронеж, 2021), а также на научных семинарах кафедры математических методов исследования операций ВГУ (2020-2023 гг.).

Достоверность и обоснованность полученных результатов обусловлена корректным использованием математических методов исследования и подтверждена результатами сравнительного анализа данных вычислительных и натурных экспериментов.

Публикации. По результатам диссертационного исследования опубликовано 11 научных работ (4 – без соавторов), в том числе 6 – в изданиях, рекомендованных ВАК РФ (из них 2 – в издании, индексируемом в международной базе данных Scopus и одно свидетельство о регистрации программы для ЭВМ). В работах, опубликованных в соавторстве и приведенных в конце автореферата, лично автором получены следующие результаты: [5] – модель частных облачных вычислительных систем с подсистемами хранения данных на основе NoSQL; [6, 9] – алгоритм динамической балансировки системы; [7, 8] – мультиагентный самонастраивающийся алгоритм репликации; [10] – сервис-ориентированная архитектура несвязанной СУБД с

модифицированной подсистемой ведения журнала; [11] – информационное и программное обеспечение человеко-машинной системы проектирования базовой структуры БД.

Структура и объем работы. Диссертационная работа состоит из введения, четырех глав, заключения, списка литературы из 194 наименований. Работа изложена на 158 страницах.

Во введении обоснована актуальность исследования, сформулированы его цель и задачи, научная новизна и практическая значимость полученных результатов, приведены сведения об апробации и внедрении работы.

В первой главе исследуются особенности разработки математического и программного обеспечения безопасного управления репликациями в масштабируемых СУБД на основе разработки мультиагентных самонастраивающихся алгоритмов репликации, и анализируется современное состояние проблемы управления ими. Отмечено, что повысить производительность и безопасность управления репликациями в масштабируемых СУБД можно путем модификации динамической балансировки системы, основанной на многокритериальном принятии решений при получении оптимальной матрицы альтернатив; а также сервис-ориентированной архитектуры СУБД с модифицированной подсистемой ведения журнала транзакций и механизмом его очистки с нулевой памятью. Потребовалась формализации данных задач, а также алгоритмизации их решения с учетом особенностей. Сформулирована цель и задачи исследования.

Вторая глава посвящена исследованию производительности облачных вычислительных сред на платформе СУБД NoSQL на основе обобщенных стохастических сетей Петри и многокритериальных оценок.

Описывается модель на основе обобщенных стохастических сетей Петри (GSPN), разработанная для представления частных облачных

вычислительных систем с подсистемами хранения данных на основе NoSQL. Модель способна оценить пропускную способность системы при различных рабочих нагрузках, доступности и простоях. Пространство состояний моделей GSPN может быть переведено в непрерывное время цепи Маркова и методы моделирования могут быть использованы для оценки показателей в качестве альтернативы генерации цепи Маркова.

Представлены экспериментальные результаты, демонстрирующие практическую осуществимость предложенной методики моделирования. Первоначально представлена проверка модели, а затем проводятся эксперименты с использованием разработанной схемы.

Далее представлен алгоритм распределения ресурсов в распределенных системах на основе двухкритериальной оценки. Каждая виртуальная машина в системе имеет свою собственную емкость. С другой стороны, процессоры имеют свои собственные атрибуты. Основная идея состоит в том, чтобы выделить и назначить правильный ресурс, который удовлетворял бы всем процессам в системе.

Предлагаемый алгоритм использует два критерия для принятия решения - это время выполнения процесса и порядок запроса/время запроса, который. Второй этап нового подхода состоит в том, чтобы проверить, находится ли система в состоянии сохранения из взаимоблокировки, если выполнение задания вызовет взаимоблокировку, то этот процесс будет ждать в очереди. Затем, когда система получит свободные ресурсы, будут обслуживаться процессы, ожидающие в очереди.

Результатом стало эффективное распределение ресурсов между задачами, а алгоритм дает системе гибкость при изменении сценариев распределения, придавая некоторым задачам приоритет больше, чем другим, вводя эффективные критерии, которые определяют, как необходимо расширять систему, чтобы использовать критерии ресурсов и

задач и комбинировать их для достижения наиболее эффективного распределения ресурсов.

Третья глава посвящена описанию механизмов проектирования самонастраиваемой репликации СУБД на основе форсированного обучения.

Рассмотрен самонастраивающийся механизм, основанный на обучении с подкреплением, для конфигурирования и регулирования в режиме реального времени гибридного промежуточного программного обеспечения репликации в паре с СУБД. Когда запросы поступают в промежуточное программное обеспечение репликации, они разбиваются на сегменты, т.е. набор логических разделов, основанных на рабочей нагрузке, которые затем назначаются отдельным репликам СУБД. Система способна исследовать конфигурацию промежуточного программного обеспечения и настроить ее на идеальную конфигурацию в режиме реального времени, не требуя затрат на перезапуск промежуточного программного обеспечения или базовой СУБД. Более того, она делает это динамично, конфигурация постоянно корректируется, чтобы отразить наилучшие значения для текущей рабочей нагрузки.

Основное внимание сконцентрировано на том, что при рассмотрении механизмов самообучения, которые способны настраивать систему промежуточного программного обеспечения репликации, можно установить, что до сих пор усилия по оптимизации рассматривали механизмы баз данных в целом и не фокусировались на конкретных внутренних компонентах, таких как контроллеры репликации. Более того, они не учитывают внешние подключаемые механизмы надежности.

Также наглядно продемонстрирована эффективность этих методов, а также основное влияние, которое регулируемые переменные настройки оказывают на общую производительность системы. Результаты подтверждают правильность подхода, подчеркивая максимальное

улучшение примерно вчетверо для некоторых из рассмотренных показателей промежуточного программного обеспечения репликации.

В главе 4 представлена архитектура распределенной защищенной многосерверной СУБД, разделяющая обработку транзакций и проектирование распределения вычислений и данных.

Рассмотрены особенности сервис-ориентированного дизайна системы в условиях слабосвязанных баз данных. Сервис-ориентированный дизайн представляет собой систему слабо связанных сервисов, которые взаимодействуют друг с другом по сети, обеспечивая гибкую конфигурацию сервиса и высокую доступность. При такой конструкции разъединения вычислений и данных служба может допускать ошибки на уровне каждого компонента: замена вычислений компонент с горячим резервированием и перенаправление компонента данных на другие узлы хранения в тех же копиях.

В предложенном архитектуре прототипа СУБД разделены компоненты в одной СУБД на два различных компонента обслуживания: сервер переднего плана, обеспечивающий обработку транзакций, и серверный компонент, гарантирующий строгую защищенность и долговечность. Сервер переднего плана отправляет журналы транзакций и получает сообщения ACK от сервера внутреннего хранилища. Сообщения ACK передают надежный порядковый номер журнала, гарантирующий долговечность журналов до определенного момента. Дополнительная логика выполнения в серверной части включает в себя восстановление обновленной страницы базы данных со старой страницы, использующая эти записи журнала. Восстановленные таким образом страницы эквивалентны обновленным страницам в интерфейсе. Когда сервер переднего плана выходит из строя, служба поддержки может немедленно продолжить обслуживание с помощью предварительно восстановленных страниц или восстанавливаемых страниц по требованию. Поэтому СУБД

необходимо только выполнить восстановление отмены. С другой стороны, сбои внутренних серверов могут быть обработаны другими внутренними серверами, которые совместно используют трафик журнала.

Сквозной дизайн реализует глобальную очередь заданий, которая используется для назначения заданий на восстановление страниц серверам. Каждый сервер использует общий менеджер буферов, который обрабатывает страницы системы ввода-вывода. С другой стороны, при проектировании "Поток-Данные" выделяются вычислительные ресурсы, такие как очередь заданий, менеджер буферов и очистка страниц для каждого сервера. Поскольку каждый сервер делится индексом страницы только с классификатором страниц способом "один писатель - несколько читателей", координация между серверами сведена к минимуму. Страницы базы данных разделяются, и задание по восстановлению новой страницы назначается выделенному исполнителю.

Рассмотрено влияние сервис-ориентированного дизайна на облачную архитектуру базы данных. Экспериментальные результаты показали, что прототип может достичь на 89% большей пропускной способности, чем базовая СУБД с рабочей нагрузкой, измеренной в sysbench.

Разработан прототип распределенной программной системы повышения производительности и безопасности репликаций облачных маршрут-нестабильных баз данных. Программная компонент реализация прошла государственную регистрацию в ФИПС.

В заключении представлены основные результаты работы.

1. Проблемы и задачи управления защищенными данными в реплицируемых СУБД

1.1. Сложности распределения ресурсов в СУБД NoSQL

1.1.1. Проблемы оценки производительности СУБД NoSQL

Производительность баз данных является важной областью исследований, поскольку хранение данных играет фундаментальную роль во многих вычислительных системах [1.1]. На протяжении многих лет исследователи также уделяли внимание оценке производительности СУБД NoSQL.

Общий подход, принятый в NoSQL, заключается в сравнении производительности с учетом различных рабочих нагрузок [1.1-1.6]. Например, в этих работах рассматривается один или несколько серверов (т.е. распределенных систем), и они обычно используют пропускную способность и время отклика в качестве интересующих показателей. Результаты показывают наиболее подходящие СУБД NoSQL для каждой конкретной рабочей нагрузки.

Что касается доступности СУБД NoSQL, то было проведено несколько работ. В [1.7] предложен анализ надежности, применяя инъекцию неисправностей путем выключения машин. Результаты экспериментов показывают влияние сбоев на возможную согласованность систем с СУБД NoSQL. Во многих работах также использовались стохастические модели для оценки вычислительных услуг, и лишь немногие модели были разработаны для системы на основе NoSQL. В [1.8] описана модель, основанная на стохастических сетях вознаграждений (SRN), для представления облачных вычислений системы, основанные на инфраструктуре как услуге (IaaS). В [1.9] авторы демонстрируют стратегию гибридного моделирования для оценки заторов системы в среде

облачных вычислений. В работе используются модели RBD и SPN для оценки доступности. В [1.10] предложен подход к моделированию, основанный на модели вознаграждений Маркова (MRM), для оценки показателей производительности систем на основе облачных вычислений. Авторы сосредоточили внимание на влиянии сбоев на производительность системы и время ремонта. В [1.11] использована цепь непрерывного времени Маркова (continuous-time Markov chain) для моделирования облачных вычислительных систем. Модели оценивают вероятность отклонения задания и среднюю задержку отклика.

В [1.12] представлена оценка трех NoSQL СУБД (Cassandra, HBase и MongoBD), а также сетевые модели массового обслуживания для планирования пропускной способности системы на основе NoSQL. В [1.13] авторы предлагают модель сети массового обслуживания для оценки производительности NoSQL СУБД, учитывая репликацию базы данных и различные уровни согласованности. Модели основаны на функциях СУБД Cassandra. В [1.14] использованы сети Петри для оценки производительности СУБД NoSQL, сосредоточив внимание на взаимосвязи между размером кластера и целостностью.

В предыдущих работах усилия были сосредоточены на оценке работоспособности, но доступность системы весьма важна, в том числе ее влияние на производительность. Далее в диссертации представлена методика, основанная на моделях GSPN, для совместной оценки производительности и доступности систем на основе NoSQL. Разработчики систем имеют дополнительный инструмент для планирования пропускной способности, учитывающий влияние сбоев на работу системы.

1.1.2. Проблема управления базами данных NoSQL в облачных средах

Объем данных, генерируемых мобильными гаджетами, датчиками и

другими вычислительными устройствами, создал концепцию больших данных.

В этом контексте системы управления базами данных NoSQL (СУБД) стали популярными благодаря большей производительности для управления большими наборами данных с помощью альтернативных моделей данных. Многие службы облачных вычислений используют СУБД NoSQL, но оценка качества обслуживания (QoS) создает дополнительные проблемы для этих систем. В главе представлен подход, основанный на обобщенных стохастических сетях Петри (GSPN) для обеспечения производительности частных облачных вычислительных сред, которые принимают СУБД NoSQL как систему хранения. Модели представлены для совместной оценки пропускной способности и доступности, которые являются важными показателями качества обслуживания. Эксперименты проводятся для демонстрации практической реализации предложенной методики.

Развитие Интернет-вещей (IoT) и популяризация мобильных устройств значительно поспособствовали созданию новой концепции, а именно Больших данных, которая характеризует скорость, объем и разнообразие данных [1.1]. В этом контексте системы управления базами данных (СУБД) NoSQL (не только SQL) стали замечательной альтернативой реляционным СУБД для решения проблем с большими данными. СУБД NoSQL обычно обладают лучшей производительностью для обработки больших наборов данных, их легко развернуть как распределенную систему, и они более эффективно управляют неструктурированными данными. Кроме того, NoSQL позволяет устанавливать различные уровни согласованности между резервными серверами [1.2].

Многие сервисы облачных вычислений используют NoSQL [1.3], но существуют дополнительные проблемы для оценки показателей качества

обслуживания (QoS), например, из-за различных рабочих нагрузок [1.4] и влияния доступности [1.5]. Некоторые работы были предложены для оценки качества обслуживания систем облачных вычислений, но существует нехватка исследований, связанная с распределенными СУБД NoSQL [1.6].

В работе предлагается совместная оценка эффективности и доступности (т.е. производительности) частных облачных вычислительных систем, использующих СУБД NoSQL в качестве подсистемы хранения. Подход основан на обобщенных стохастических сетях Петри (GSPN) и моделях, позволяющих оценивать пропускную способность и доступность в качестве важных показателей качества обслуживания для оценки систем распределенного хранения. Эксперименты, основанные на Yahoo! - Облачном Сервисе Benchmark (YCSB), проводятся для иллюстрации практической применимости предлагаемых моделей.

1.1.3. Проблемы распределения ресурсов в распределенных системах

В последнее время использование высокопроизводительных вычислений значительно возросло. Хотя стоимость предоставления таких систем относительно высока, поэтому единственным решением является совместное использование этих ресурсов и параллельное их использование несколькими процессами, что обеспечивает высокопроизводительную обработку при низких затратах и особенно более эффективно [1.30]. Современная среда распределенных систем обеспечивает совместное использование ресурсов, кроме того, в соответствии с архитектурой этих систем содержит несколько виртуальных машин или узлов, каждый из которых имеет свои локальные процессоры и локальную память, которые повышают пропускную способность всей системы.

Наиболее важными преимуществами использования распределенных систем являются предоставление задачам возможности доступа к ресурсам

системы, надежность, отказоустойчивость, масштабируемость и высокая скорость работы [1.31], и это приведет к снижению затрат. В распределенной среде можно выполнять несколько процессов, используя различные доступные виртуальные машины в системе. Наиболее яркими примерами являются сети и облачные вычисления, которые могут предоставлять эти услуги: в инфраструктуре (IaaS), которые представляют инфраструктуру как услугу (PaaS), которые относятся к платформе как к услуге и программному обеспечению как к услуге (SaaS). Принцип виртуализации явно используется в (IaaS), когда система совместно использует виртуальные машины. Таким образом, управление ресурсами - это управление потоком данных заданий и выделение необходимого ресурса для каждой задачи [1.32].

Распределение ресурсов (RA) в значительной степени зависит от стратегии, которая будет управлять использованием системы, особенно когда огромное количество задач ожидает обработки в очереди [1.33].

Поэтому проблема разработки эффективной стратегии распределения ресурсов является актуальной задачей.

В [1.33, 1.34] было изучено планирование распределения ресурсов. Однако авторы работ предлагают использовать в качестве критерия оптимальности только один критерий. Таким критерием может быть стоимость ресурсов или максимизация качества обслуживания (QoS). Вопросы разработки эффективных алгоритмов планирования ресурсов возможны с учетом нескольких критериев

Предлагаемый алгоритм будет сосредоточен на оптимизации планирования задач путем использования атрибута этих задач в качестве критерия. Алгоритм пытается достичь эффективного прогнозирования, и лучшим методом в этой ситуации является заставить систему автоматически определять планирование задач в соответствии с двумя критериями выбора порядка обслуживания (время обслуживания и

важность задач) с помощью многокритериального принятия решений (MCDM) [1.35]. В этом случае многие процессы представляют альтернативы, которые система должна выбрать, какие из них будут обслуживаться в первую очередь, и переставить в очередь для обслуживания виртуальными машинами, представляющими ресурсы.

Проблема распределения ресурсов (RAP) является основной проблемой, для решения которой требуется эффективная стратегия во всех совместно используемых системах. Основная цель задач планирования состоит в том, чтобы определить ресурсы для каждого процесса, который должен быть выполнен. Таким образом, эффективность стратегии планирования заключается в минимизации времени выполнения задач и достижении наилучшего распределения ресурсов в сочетании с QoS. В распределенной системе ресурсы распределяются между задачами (процессор, объем памяти, программное обеспечение и т.д.), как показано на рис. 1.1.

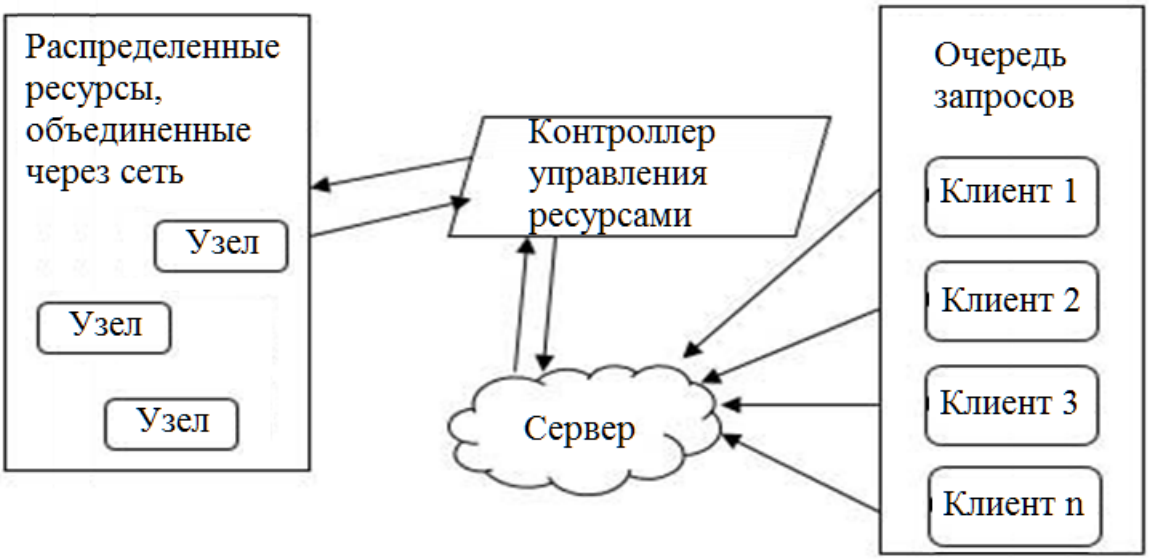


Рис. 1.1. Основная структура механизма распределения ресурсов в распределенной системе

Проблема планирования может быть просто описана n+1 процессами или задачами, запрашивающими ресурсы (m – количество виртуальных

машин) из системы, в результате ограничения ресурсов и отсутствия общих задач [1.31], метод планирования задач должен минимизировать разрыв между временем запроса и время отправки, предотвращение сбоев, например, взаимоблокировки, и учет назойливости задачи.

В соответствии с атрибутом распределенной системы предлагаемый алгоритм будет зависеть от метода многокритериального принятия решений (MCDM) для распределения ресурсов, наиболее эффективным и гибким методом ранжирования является Метод организации ранжирования предпочтений для оценок обогащения (PROMETHEE II) [1.36]. Этот метод может решить проблему принятия решений (DM) для конечного набора альтернатив, используя попарное сравнение для получения ранжирования в зависимости от значений каждого критерия. Поэтому в предлагаемом алгоритме мы использовали этот метод для решения задачи (DM) для огромного конечного числа задач, которые могут запрашивать ресурсы распределенной системы.

В целом, моделирование проблемы распределения ресурсов в распределенных системах должно обеспечивать среду, которая имеет возможность совместного использования ресурсов. Эта способность позволяет запрашивающему процессу использовать аппаратные и программные возможности всей системы при выполнении с высокой производительностью [1.36]. Кроме того, согласно концепциям потока данных в распределенных системах, управление ресурсами повысит эффективность среды выполнения, а также механизм планирования задач закрыт для оптимального распределения ресурсов.

1.2. Задачи обеспечения механизмов репликации, обеспечения отказоустойчивости и высокой доступности в распределенных СУБД

В настоящее время существует множество СУБД, каждая из которых

ориентирована на определенный тип рабочей нагрузки, а именно: оперативная обработка транзакций (OLTP), оперативная аналитическая обработка (OLAP) и даже более новые гибридные рабочие нагрузки по транзакционной и аналитической обработке (HTAP). Несмотря на то, что существуют рекомендации по оценке и настройке СУБД, конфигурации не являются универсальными, и каждый поставщик решает, какие настраиваемые конфигурации следует предоставлять. Кроме того, типы рабочей нагрузки очень различаются, чтобы обеспечить общую конфигурацию в разных системах. Это делается для того, чтобы оптимизация целевой операции OLTP существенно снизила производительность OLAP и наоборот [1.16]. Поскольку тип конфигураций доступных для настройки увеличивается и проблемы, связанные с данным типом рабочей нагрузки различны для каждого поставщика, администратору базы данных ответственен за оценку и точную настройку рабочей нагрузки в конкретной СУБД.

Это особенно верно для механизмов репликации и обеспечения отказоустойчивости и высокой доступности, поскольку такие функции обычно тесно связаны с тем, как каждая СУБД обрабатывает данные, особенно в системах OLTP. Системы баз данных обычно обеспечивают отказоустойчивость за счет снижения пропускной способности репликации, когда алгоритмы сложно развертываются как часть ядра СУБД. При сопряжении подсистемы СУБД с обеспечением гарантий отказоустойчивости через внешние системы промежуточного программного обеспечения количество рассматриваемых настраиваемых конфигураций увеличивается. Из преимуществ, связанных с разделением, следует отметить модульность и подключаемость. Это достигается за счет более высокой сложности и возможного разделения между проектированием промежуточного программного обеспечения репликации и их воздействием на рассматриваемую базовую СУБД. Кроме того,

логическое разделение обычно достигается с помощью стандартных интерфейсов, таких как Java DataBase Connector (JDBC), что также вызывает новые проблемы.

Важная работа, как правило, сосредоточена на обеспечении оптимизации физического развертывания и компоновки данных систем баз данных. Связь с физическим расположением данных позволяет системам самонастройки ускорить создание индексов [1.17], разделов данных [1.18] или материализованных представлений [1.19]. Однако эти подходы не в состоянии охватить детали, лежащие в основе СУБД, где корректировка конкретных требований к настройке влияет на внутренние компоненты СУБД. Большинство продуктов баз данных, таких как IBM BD2, Microsoft SQL Server или MySQL, включают в себя инструменты для настройки соответствующих систем в отношении производительности сервера [1.20, 1.21]. Вообще говоря, эти системы тестируют альтернативные конфигурации при развертывании вне производства и оценивают их результаты. Однако это решение зависит от усмотрения администратора базы данных при выборе стратегий для применения и того, какие последующие действия предпринимаются в СУБД.

Описанный набор инструментов также (косвенно) охватывает настраиваемые критерии, которые строго касаются настройки методов репликации, как часть настраиваемых переменных конфигурации, которые доступны в каждой системе. Однако учет настраиваемых переменных, как правило, отсутствует [1.16], хотя некоторые системы специализируются на выборе наиболее значимых конфигураций [1.22]. Тем не менее, обычно это независимый процесс.

Хотя стратегии обучения с подкреплением не являются новинкой для групп, интересующихся машинным и автономным обучением, в настоящее время они являются модными методами, которые можно применять в различных областях вычислений. Этот класс методов охватывает агента,

который пытается получить изменения состояния в своей соответствующей среде, стремясь максимизировать баланс долгосрочной и краткосрочной отдачи от функции вознаграждения [1.23]. В частности, в области систем баз данных такие стратегии начали применяться во внутренних системах СУБД, чтобы помочь оптимизаторам запросов превзойти базовые стратегии оптимизации для упорядочения соединений [1.24] и общего планирования запросов [1.25]. Результатом обычно является последовательный набор решений, основанных на цепочках решений Маркова [1.26, 1.27]. Рассмотренный метод RL не изменяет критерий оптимизации, а скорее то, как проводится процесс разведки.

Эти методы позволяют агенту искать действия на этапе обучения, когда он намеренно предпринимает действия, чтобы узнать, как реагирует окружающая среда, или приблизиться к цели оптимизации. Более того, эти методы обычно основаны на эвристических методах исследования, основанных на статистических данных, собранных из окружающей среды.

Недавно были представлены решения на основе RL, специфичные для самостоятельной настройки базы данных с одним экземпляром, они используют либо метрики, предоставляемые базой данных [1.28], либо информацию из оптимизатора запросов СУБД и тип запросов, которые составляют рабочую нагрузку базы данных [1.29]. Далее показано, что это также возможно на уровне промежуточного программного обеспечения репликации. Кроме того, вводится идея динамической настройки, означающая, что переменные настройки рассматриваются как постоянно изменяющиеся, адаптируя их значения в режиме реального времени к изменениям рабочей нагрузки.

Будет показано, что хотя глубокое Q-обучение не может работать с пространствами непрерывных действий без их выборки, в отличие от некоторых более поздних алгоритмов, таких как Deep Deterministic Policy Gradient (DDPG), ему все же удается достичь очень хороших результатов.

Использование глубокого Q-обучения по сравнению с DDPG может быть выгодным в условиях ограниченных ресурсов, поскольку в нем используется только одна нейронная сеть вместо двух (одна для актера и одна для критика).

1.3. Программная архитектура систем облачных вычислений

1.3.1. Облачные сервисы и центры обработки данных

С распространением облачных вычислений произошли большие изменения в программной архитектуре сервисов, почти все услуги можно адаптировать с помощью облака. Отличительной чертой облачного сервиса является уменьшенная стоимость управления ресурсами, которая в противном случае может потребовать детализации от первоначальной сервисной настройки до долгосрочного сервисного обслуживания. Фактически, веб-приложения и сервисы хранения данных широко используются в облаке, а СУБД становится все более популярным и в облаке. Сервисы облачных СУБД, такие как Amazon AuroraDB, Google Cloud SQL и база данных Microsoft Azure SQL предоставляют высокую надежность сервисов, сервисное обслуживание и отказоустойчивость для роста сервисов на основе облачной инфраструктуры.

Однако облачные сервисы обычно работают в центрах обработки данных. Предоставляя большое количество стандартных серверов, подключенных через высокоскоростные сети, все они уязвимы для различных типов сбоев. Это неизбежно приводит архитекторам программного обеспечения уделять первоочередное внимание высокой доступностью сервисов при планировании развертывания сервисов в облаке. Большинство поставщиков облачных услуг предлагают множество практических решений, реализующих отказоустойчивость запуская экземпляры сервисов и запускать их в случае сбоев. Например, типичное

решение для аварийного переключения - использование метода горячего резервирования, в котором резервный экземпляр при обнаружении отказавшего экземпляра берет на себя обслуживание. Состояние услуги может быть возобновлено только после того, как все состояния были восстановлены сразу после сбоя, а службы без сохранения состояния можно легко перезапустить после сбоя. Кроме того, СУБД в облачном сервисе требуют более сложную поддержку восстановления.

Используется либо копирование данных, либо независимая организация, чтобы избежать этого восстановления на уровне приложения. В отличие от монолитной организации, которая обычно используется в традиционных СУБД, СУБД в изолированной организации могут допускать отказы компонентов, тем самым повышая доступность услуг. Для анализа проектных решений для распределенной организации СУБД, доработаны некоторые компоненты в реальной СУБД, чтобы отделить как ведение журнала, так и процедуру очистки страниц от экземпляра СУБД. У этой независимой организации есть еще одна сильная сторона для облачных СУБД. Во-первых, асинхронная транзакция ведения журнала может скрыть увеличенную задержку фиксации транзакций и еще больше повысить производительность обработки транзакций. Во-вторых, восстановление страницы вместо ее сброса может уменьшить сетевой трафик, и этот внешний процесс для транзакции обработки предназначен для минимизации времени восстановления базы данных. Третье, использование параллельного восстановления страниц. Журнал транзакций в стиле OBHA и различные схемы назначения этих заданий могут организовать эффективную конвейерную архитектуру для крупномасштабной системы обработки транзакций.

Реализован прототип распределенной организации на основе MariaDB, одного из популярных вариантов MySQL. Оценки проводились на локальных твердотельных накопителях (SSD) с тестом sysbench, и он

продемонстрировал, что асинхронное ведение журнала прототипа работает почти на 89% лучше, чем в оригинальной MariaDB. В то же время параллельное восстановление страниц обеспечивает стабильную производительность системы при минимальных требованиях к системным ресурсам. Также изучено использование ресурсов для этого дополнительного процесса и эффективность восстановления страниц для параллельной работы восстановления страниц. Наконец, проведено сравнение двух различных схем задания по восстановлению страниц: поток к работе и поток к данным. Подход "поток-данные" обеспечивает более высокий уровень масштабируемости, чем подход "поток-работа", а затем обеспечивает более высокую производительность.

1.3.2. Облачные приложения

Поставщики облачных услуг настраивают фермы серверов как взаимосвязанные серверы с сетевыми подключениями в центре обработки данных и предлагают абстрактную среду обслуживания для арендаторов облачных услуг. Основные преимущества этого облачного сервиса можно обобщить двумя способами. Во-первых, упрощенная конфигурация службы на этапе установки или развертывания может сэкономить много времени и денег для арендаторов услуг. Во-вторых, простота обслуживания упрощает усилия по достижению эластичной масштабируемости и высокой доступности. Автоматический механизм аварийного переключения заменяет неисправный экземпляр с новым экземпляром и копирует данные, чтобы обеспечить надежную защиту от потери данных в случае сбоев. Однако восстановление на уровне приложений по-прежнему остается за пользователями.

1.3.3. Восстановление базы данных

Разрыв в производительности между памятью и хранилищем

приводит СУБД, предназначенную для эффективного ввода-вывода, в базовое хранилище блоков. Традиционные архитектуры баз данных используют страницы логических данных фиксированного размера из нескольких блоков и управляют изменениями данных для эффективного ввода-вывода и шаблона доступа с помощью абстракции транзакции. Любые изменения данных объединяются в файлы журнала только для добавления и в конечном итоге сохраняются в хранилище, вместо того чтобы записывать всю страницу данных при каждом обновлении. Для восстановления базы данных требуется восстановить обновленные страницы со старыми страницами данных и файлами журналов из хранилища.

Восстановление базы данных происходит с помощью журналов, хранящихся на сетевых устройствах хранения, обычно задерживается из-за ограниченной пропускной способности и задержки в сети. На рис. 1.2 показано влияние типов хранилища на время восстановления.

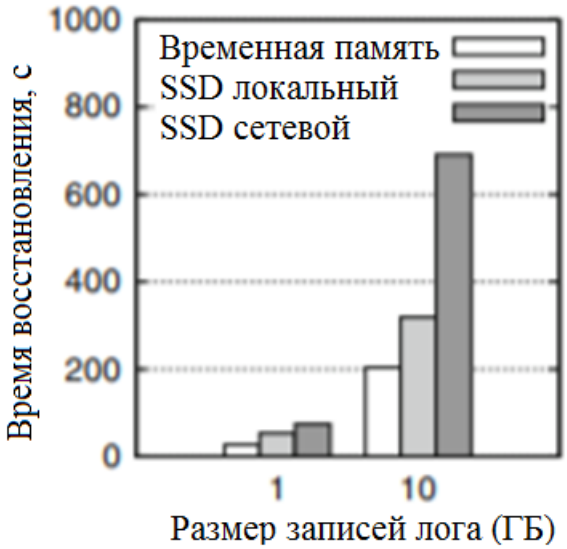


Рис. 1.2. Время восстановления базы данных для различных типов хранилищ

Извлечение страниц и файлов журналов из сетевого хранилища, мешающие друг другу в общем сетевом трафике, негативно влияют на

производительность процедуры восстановления. В дополнение к типу хранилища формат журнала также влияет на время восстановления.

1.3.4. Ведение журнала базы данных и установка контрольных точек

Ведение журнала может быть одного из двух типов — физическое ведение журнала и логическое ведение журнала. Поскольку физическое (значение) ведение журнала, которое записывает как старые, так и новые значения на физической странице, не раскрывает зависимости данных на других страницах или дополнительное вычисление нового значения, оно может создавать страницу данных только с записями журнала, относящимися к странице, а не к другим страницам.

Но, поскольку физическое ведение журнала требует, чтобы оно содержало как старые, так и новые значения, и оно генерирует большое количество журналов в пространстве хранения. С другой стороны, логическое ведение журнала может уменьшить количество записей, записывая только команду (или инструкцию), а не физические изменения, но оно может увеличить время восстановления, поскольку оно должно запускать команду (или инструкцию) в журнале для создания страницы во время восстановления. ОВЕН, широко используемый метод восстановления, использует физиологическое ведение журнала (т.е. физическое повторение и логическое отмена) для балансировки ведения журнала и производительности восстановления [1.40].

Большое количество записей или истории обновления данных может затруднить процедуру восстановления, и СУБД периодически очищает страницы данных в памяти (т. е. контрольную точку). На рис. 1.3 показан общий объем ввода-вывода при ведении журнала и очистке страниц. Для очистки страниц требуется значительно больший объем ввода-вывода, чем для журнала, и как размер страницы, так и рабочая нагрузка влияют на

общее количество очистки страниц.

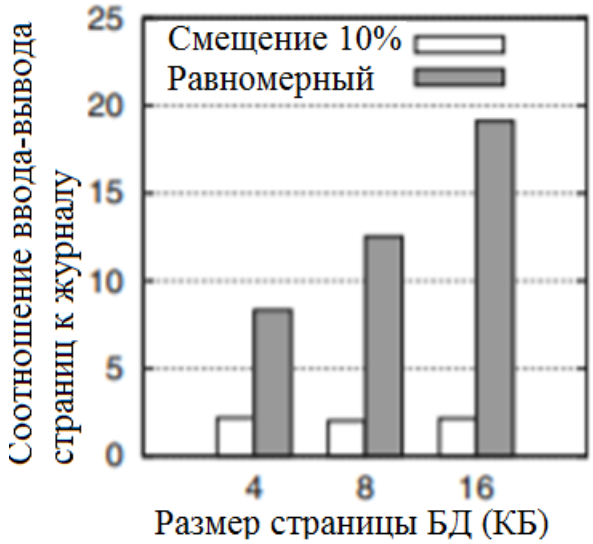


Рис. 1.3. Соотношение ввода-вывода двух страниц к журналу при запуске теста sysbench

Разъединенная организация обычно используется в сервис-ориентированной архитектуре (SOA). Потенциал этой архитектуры для гибкого масштабирования и высокой доступности позволяет продуктам СУБД использовать эти функции [1.41]. Предложена аналогичная архитектура СУБД, но эта архитектура состоит из многоуровневых и иерархических компонентов [1.42]. Под влиянием этой многоуровневой архитектуры несколько проектов СУБД предполагают, что модульная архитектура может снизить затраты на разработку и обслуживание в монолитной организации СУБД [1.43-1.45]. Однако высокая производительность также является важным требования к СУБД, и некоторые компоненты должны быть тесно интегрированы для повышения эффективности.

Прототип SO-СУБД отправляет журналы транзакций по сети. Очистка журналов по сети является распространенным методом для реализаций СУБД с основной памятью для обеспечения надежности транзакций. Поскольку СУБД с основной памятью обычно предназначены

для высокопроизводительных механизмов транзакций, ведение журнала может снизить производительность системы в этой системе. Hekaton [1.46] предназначен для снижения затрат на ведение журнала за счет нескольких оптимизаций, таких как ведение журнала только для повторного выполнения, несколько потоков журналов и групповая фиксация. VoltDB [1.47] использует ведение журнала команд вместо естественного ведения журнала, варианты ARIES [1.48] используют, чтобы уменьшить как вычислительные издержки, так и общий объем ведения журнала во время выполнения [1.49]. Этот тип схем ведения журнала повышает производительность во время выполнения с производительностью восстановления. Хотя различные средства поддержки высокой доступности требуют менее частых процедур восстановления, ведение журнала команд все еще требует большего времени восстановления с дополнительной обработкой для идентификации затронутых данных и ограниченным параллелизмом восстановления данных.

Прототип SO-СУБД использует формат журнала физического повторного входа в ARIES для параллельного восстановления страниц. Аналогичный подход был предложен, чтобы избежать длительного времени восстановления. Алгоритм мгновенного восстановления также использует формат физического журнала и строит индекс восстановления страницы, содержащий упорядоченный по времени список журналов транзакций для каждой страницы на этапе анализа процедуры восстановления [1.50]. Используя этот индекс восстановления, система может восстанавливать страницы по требованию и обновлять их даже во время восстановления системы. Этот алгоритм может работать с внешний процесс архивирования журналов. Этот процесс объединяет журналы в контрольную точку параллельно, тем самым уменьшая разрыв между последней контрольной точкой и текущими записями журнала. В то время как алгоритм мгновенного восстановления разработан в случае сбоя СУБД,

параллельное восстановление страниц в прототипе предназначено для предотвращения промывки страниц по сети, которая требует значительной пропускной способности сети.

Прототип SO-СУБД применяет журналы транзакций к страницам почти мгновенно при фиксации транзакции. Аналогичный подход предполагает, что изменения данных публикуются при каждой фиксации транзакции. Алгоритм Meld [1.51] использует дополнительный процесс для выполнения транзакций для создания обновленных древовидных данных с журналами, одновременно добавляемыми исполнителями транзакций. Алгоритм Meld изначально был разработан для высокопроизводительного механизма транзакций с данными общего дерева или индексами баз данных. В то время как исполнители параллельных транзакций изменяют данные и добавляют их в общий журнал, один объединитель сериализует изменения данных и обнаруживает конфликты данных во время одновременных транзакций. Поскольку объединитель не должен ограничивать общесистемную производительность обработки транзакций, этот алгоритм предназначен для записи небольшого количества журналов (т.е. только последнего значения и метаданных для обнаружения конфликтов). Любые неконфликтные журналы транзакций постепенно объединяются в дерево путем арбитража объединителя, таким образом восстанавливая актуальное состояние дерева. Вместо того, чтобы использовать эту единственную оптимизацию процесса, мы расширяем внешний процесс на группу рабочие потоки, которые параллельно объединяют журналы в индекс базы данных.

Представлен другой подход к публикации изменений данных с аппаратной поддержкой. Новые технологии хранения данных, такие как NVRAM, обеспечивают доступ к данным с адресом в байтах, и это не показывает существенной разницы в производительности между

случайным и последовательным доступом к данным. MARS [1.52] изменяет ARIES для использования этих новых устройств хранения данных, а также предоставляет дополнительную аппаратную поддержку для атомарного обновления нескольких данных. В MARS один арбитр, аналогичный алгоритму Meld, упорядочивает последовательность фиксаций транзакций и применяет изменения данных в локальном буфере журнала транзакций к устройствам хранения.

Высокопроизводительные системы обработки транзакций часто используют дизайн "ничего общего", который сводит к минимуму координацию при одновременном обновлении общих данных. H-store [1.53] организует независимые логические экземпляры базы данных с предварительно разъединенными данными, и каждая транзакция может быть обработана без использования блокировки. DORA [1.54] предлагает назначение потоков данных, а не назначение потоков транзакций, чтобы исключить координацию обновлений. Схема разъединения физических данных трудно предсказать схему доступа к данным при рабочих нагрузках OLTP, и трудно повторно разделить данные при изменении асимметрии данных. Несколько подходов используют логический менеджер секционирования для обеспечения гибкости секционирования [1.55, 1.56]. Хотя серверная часть в SO-СУБД не требует координации обновлений страниц с использованием физического разбиения данных на страницы истории обновлений, она по-прежнему требует гибкости разделов.

1.4. Постановка задач работы

Целью работы является разработка математического и программного обеспечения безопасного управления репликациями в масштабируемых СУБД на основе мультиагентного

самонастраивающегося алгоритма.

Для достижения цели необходимо решить следующие **задачи**:

1. Разработать модель частных облачных вычислительных систем с NoSQL-подсистемами хранения данных на основе обобщенных стохастических сетей Петри.

2. Создать алгоритм динамической балансировки системы основанный на многокритериальном принятии решений при получении оптимальной матрицы альтернатив.

3. Предложить мультиагентный самонастраивающийся алгоритм репликации, основанный на обучении с подкреплением с использованием монитора метрик, диспетчера обновлений и агентов обучения с подкреплением.

4. Создать сервис-ориентированную архитектуру несвязанной СУБД с модифицированной подсистемой ведения журнала транзакций и механизмом его очистки с нулевой памятью.

5. Разработать архитектуру человеко-машинной системы проектирования базовой структуры БД с интерактивным характером выбора оптимальных характеристик межсистемного взаимодействия.

Список источников к главе 1

1.1. Osman R., Knottenbelt W.J. Database system performance evaluation models: A survey// Performance Evaluation, vol. 69, no. 10, pp. 471 - 493, 2012.

1.2. Abramova V., Bernardino J. NoSQL databases: MongoDB vs Cassandra// International C* Conference on Computer Science and Software Engineering, ser. C3S2E ’13. ACM, 2013, pp. 14-22.

1.3. Dede E. et al. Performance evaluation of a mongoDB and hadoop platform for scientific data analysis// 4th ACM Workshop on Scientific Cloud Computing, ser. Science Cloud ’13. ACM, 2013.

1.4. Klein J. et al. Performance evaluation of noSQL databases: A case study// 1st Workshop on Performance Analysis of Big Data Systems, ser. PABS ’15. ACM, 2015.

1.5. Tang E., Fan Y. Performance comparison between five noSQL databases// 2016 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 105-109, 2016.

1.6. Gomes C., Tavares E.A.G., de Nogueira M.O. Jr. Energy consumption evaluation of noSQL DBmss// 15 WPerformance - Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2016.

1.7. Ventura L., Antunes N. Experimental assessment of noSQL databases dependability// 2016 12th European Dependable Computing Conference (EDCC), 2016.

1.8. Bruneo D. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems// IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 560-569, 2014.

1.9. de Lima Q.V. et al. Performability evaluation of emergency call center// Performance Evaluation, vol. 80, pp. 27 - 42, 2014.

1.11. Kirsal Y. et al. Analytical modeling and performability analysis for

cloud computing using queuing system// in 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015.

1.12. Ghosh R. et al. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach// 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, 2010.

1.13. Gandini A. et al. Performance evaluation of noSQL databases// Computer Performance Engineering. Springer International Publishing, 2014.

1.14. Dipietro S., Casale G., Serazzi G. A queuing network model for performance prediction of Apache Cassandra// 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), 2017.

1.15. Osman R., Piazzolla P. Modeling replication in noSQL datastores// Quantitative Evaluation of Systems. Springer International Publishing, 2014.

1.16. Duan S., Thummala V., Babu S. Tuning database configuration parameters with ituned// Proc. VLDB Endow., 2009, vol. 2(1), pp. 1246-1257.

1.17. Valentin G., Zuliani M., Zilio D.C., Lohman G., Skelley A. Db2 advisor: an optimizer smart enough to recommend its own indexes// Proc. of 16th Int. Conf. on Data Engineering (Cat. no. 00CB37073), 2000, pp. 101-110.

1.18. Curino C., Jones E., Zhang Y., Madden S. Schism: a workload-driven approach to database replication and partitioning// Proc. VLDB Endow., 2010, vol. 3(1-2), pp. 48-57.

1.19. Agrawal S., Chaudhuri S., Narasayya V.R. Automated selection of materialized views and indexes in SQL databases// VLDB, 2000, pp. 496-505.

1.20. Dias K., Ramacher M., Shaft U., Venkataramani V., Wood G. Automatic performance diagnosis and tuning in Oracle// CIDR, 2005, pp. 84-94.

1.21. Schiefer K.B., Valentin G. Db2 universal database performance tuning// IEEE Data Eng. Bull., 1999, vol. 22(2), pp. 12-19.

1.22. Sullivan D.G., Seltzer M.I., Pfeffer A. Using Probabilistic Reasoning to Automate Software Tuning, vol. 32. ACM, New York, 2004.

1.23. Garcıa J., Fernandez F. A comprehensive survey on safe reinforcement learning// J. Mach. Learn. Res., 2015, vol. 16(1), pp. 1437-1480.

1.24. Trummer I., Moseley S., Maram D., Jo S., Antonakakis J. Skinnerdb: regretbounded query evaluation via reinforcement learning// Proc. VLDB Endow., 2018, vol. 11(12), pp. 2074-2077.

1.25. Marcus R., Papaemmanouil O. Deep reinforcement learning for join order enumeration// Proc. of the First Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018, pp. 3:1-3:4.

1.26. Morff A.R., Paz D.R., Hing M.M., Gonzalez L.M.G. A reinforcement learning solution for allocating replicated fragments in a distributed database// Computacion y Sistemas, 2007, vol. 11(2), pp. 117-128.

1.27. Puterman M.L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken, 2014.

1.28. Zhang J. An end-to-end automatic cloud database tuning system using deep reinforcement learning// Proc. of the 2019 Int. Conf. on Management of Data, 2019, pp. 415-432.

1.29. Li G., Zhou X., Li S., Gao B. Qtune: a query-aware database tuning system with deep reinforcement learning// Proc. VLDB Endow, 2019, vol. 12(12), pp. 2118-2130.

1.30. Thamsen L, Verbitskiy I, Beilharz J, Renner T, Polze A, Kao O. Ellis: Dynamically Scaling Distributed Dataflows to Meet Runtime Targets. Proc. Int/ Conf. Cloud Comput Technol Sci CloudCom. 2017;(37)146–153.

1.31. Silberschatz A., P. Baer Galvin, Gagne G. Operating Systems Concepts. New York: John_Wiley_&_Sons. 2008;(8):741-750.

1.32. Thamsen L., Renner T, Kao O. Continuously Improving the Resource Utilization of Iterative Parallel Dataflows. Proceedings of the 6th International Workshop on Big Data and Cloud Performance, ser. DCPerf 2016.

IEEE. 2016;1(4):1–6.

1.33. Castillo G., Rouskas N., Harfoush K. Efficient QoS resource management for heterogeneous Grids. 22nd. IEEE International Parallel and Distributed Processing Symposium (IPDPS'08), Miami, Florida, US. 2008;1-15.

1.34. Jiang H., Ni T. PB-FCFS-a task scheduling algorithm based on FCFS and backfilling strategy for grid computing. Proceedings of Joint Conferences on Pervasive Computing (JCPC). 2009; 507- 510.

1.35. Triantaphyllou E. Multi-Criteria Decision-Making Methods in Multi-Criteria Decision-Making Methods, a Comparative Study. Applied Optimization. 2000; 44(1): 5–21.

1.36. Chen L., Xu Z., Wang H., Liu S. An ordered clustering algorithm based on K-means and the PROMETHEE method. International Journal of Machine Learning and Cybernetics. 2018; 9(6):917-926.

1.37. Chakraborty S., Yeh C. H. A Simulation Based Comparative Study of Normalization Procedures in Maldistributed Decision Making. Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases.2007; 102–109.

1.38. Mareschal B., Brans Jean-Pierre. PROMETHEE methods. International Series in Operations Research and Management Science.2014;78(2):163-195.

1.39. Yu L., Chen L., Cai Z., Shen H., Liang Y., Pan Y. Stochastic Load Balancing for Virtual Resource Management in Datacenters. IEEE Transactions on Cloud Computing. 2018; 8(2):459–472.

1.40. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst. 17(1), 94–162 (1992)

1.41. Campbell, D.: Service oriented database architecture: App serverlite? In: Proceedings of the 2005 ACM SIGMOD international conference on Management of Data, pp. 857–862. ACM (2005)

1.42. Harder, T.: Dbms architecture - the layer model and its evolution. Datenbank-Spektrum 13, 45–57 (2005)

1.43. Tok, W.H., Bressan, S.: Dbnet: A service-oriented database architecture. In: Proceedings of the 17th International Workshop on Database and Expert Systems Applications, DEXA'06. pp. 727–731. IEEE (2006)

1.44. Subasu, I., Ziegler, P., Dittrich, K.R.: Towards service-based data management systems. In: Proceedings of the Workshop on Datenbanksysteme in Business, Technologie und Web (BTW 2007), pp. 3–86,130. Citeseer (2007)

1.45. Irmert, F., Daum, M., Meyer-Wegener, K.: A new approach to modular database systems. In: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management, pp. 40–44. ACM (2008)

1.46. Diaconu, C., Freedman, C., Ismert, E., Larson, P.A., Mittal, P., Stonecipher, R., Verma, N., Zwilling, M.: Hekaton: Sql server's memory-optimized oltp engine. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1243–1254. ACM (2013)

1.47. Stonebraker, M., Weisberg, A.: The voltdb main memory dbms. IEEE Data Eng. Bull. 36(2), 21–27 (2013)

1.48. Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P.: Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst. 17(1), 94–162 (1992)

1.49. Malviya, N., Weisberg, A., Madden, S., Stonebraker, M.: Rethinking main memory OLTP recovery. In: Proceedings of the 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 604–615. IEEE (2014)

1.50. Graefe, G., Guy, W., Sauer, C.: Instant recovery with write-ahead logging: page repair, system restart, and media restore. Synth. Lect. Data Manag. 6(5), 1–85 (2014)

1.51. Reid, C., Bernstein, P., Wu, M., Yuan, X.: Optimistic concurrency

control by melding trees. Proc. VLDB Endow. 4(11), 944–955 (2011)

1.52. Coburn, J., Bunker, T., Schwarz, M., Gupta, R., Swanson, S.: From aries to mars: Transaction support for next-generation, solid-state drives. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp. 197–212. ACM (2013)

1.53. Stonebraker, M., Madden, S., Abadi, D.J., Harizopoulos, S., Hachem, N., Helland, P.: The end of an architectural era: (it's time for a complete rewrite). In: Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1150–1160. VLDB Endowment (2007)

1.54. Pandis, I., Johnson, R., Hardavellas, N., Ailamaki, A.: Data-oriented transaction execution. Proc. VLDB Endow. 3(1–2), 928–939 (2010)

1.55. Pandis, I., To¨zu¨n, P., Johnson, R., Ailamaki, A.: Plp: page latchfree shared-everything oltp. Proc. VLDB Endow. 4(10), 610–621 (2011)

1.56. Pavlo, A., Curino, C., Zdonik, S.: Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 61–72. ACM (2012)

2. Исследование производительности облачных вычислительных сред на платформе СУБД NoSQL на основе обобщенных стохастических сетей Петри и многокритериальных оценок

2.1. Особенности СУБД NoSQL

Базы данных NoSQL стали альтернативой реляционным СУБД благодаря лучшей производительности для работы с большими наборами данных и альтернативными моделями данных, подходящих для неструктурированных данных. СУБД NoSQL не используют свойства ACID [2.2], но они используют базовую концепцию. Без потери общности последнее фокусируется на производительности за счет использования возможной согласованности [2.19].

Разделение данных также является важной особенностью NoSQL СУБД, поскольку они обычно обеспечивают большую масштабируемость, чем реляционные аналоги. Конечная согласованность позволяет разделить набор данных на непересекающиеся разделы и развернуть их на нескольких серверах, не беспокоясь о согласованности данных. Лучшая доступность достигается за счет репликации данных и наличия механизмов восстановления после сбоев, которые также облегчаются благодаря возможной совместимости [2.19].

Некоторые СУБД NoSQL (например, MongoDB) обеспечивают горизонтальное масштабирование с использованием вариабельного метода, а именно сегментирования. Каждый сегмент содержит фрагмент набора данных и несколько узлов (т. е. серверную машину) с теми же данными, а именно набором реплик. Основной узел отвечает за все операции записи и чтения. Если первичный узел не работает из-за сбоя, используется запасной узел.

2.2. Применение обобщенных стохастических сетей Петри при моделировании системы

Сети Петри (PN) [2.20] представляют собой семейство формализмов, подходящих для представления многих типов систем (например, облачных вычислений), поскольку они естественным образом представляют системы с параллелизмом, синхронизацией и механизмами связи, а также временными задержками. Без потери общности сеть Петри представляет собой двудольный ориентированный граф с 2 типами узлов: местами и переходами. Места (представленные кружками) обозначают локальные состояния, а переходы (изображенные прямоугольниками) представляют действия. Дуги - это направленные ребра, которые соединяют места с переходами и наоборот. Места могут содержать маркеры (маленькие заполненные кружочки), которые представляют состояние (т. е. маркировку) PN. Дуга ингибитора - это особый тип дуги, который может проверять доступность токенов в определенном месте. Такая дуга изображена с небольшим белым кружком на одном краю, вместо стрелки.

В работе используются обобщенные стохастические сети Петри (GSPN) [2.21], важное расширение PN, которое позволяет связать экспоненциальное распределение с временными переходами (представленными белыми прямоугольниками) или нулевыми задержками с немедленными переходами (изображенными в виде тонких черных прямоугольников). Кроме того, временные переходы могут иметь разные степени параллелизма: семантика одного сервера (ss) или семантика бесконечного сервера (is). В целом, is принимается для представления параллельных передач, учитывая, что скорость передачи линейно увеличивается в соответствии со степенью, обеспечивающей переход. В ss скорость передачи поддерживается постоянной. Поведение GSPN представлено игрой с токенами (т.е. изменением маркировки GSPN) в том

смысле, что токены потребляются и создаются в соответствии с запусками перехода.

Пространство состояний моделей GSPN может быть переведено в непрерывное время цепи Маркова (CTMC) [2.22] и методы моделирования могут быть использованы для оценки показателей в качестве альтернативы генерации цепи Маркова.

2.3. Системное моделирование представления облачных вычислительных систем с подсистемами хранения данных на основе NoSQL

В этом разделе описывается модель GSPN, разработанная для представления частных облачных вычислительных систем с подсистемами хранения данных на основе NoSQL. Модель способна оценить пропускную способность системы при различных рабочих нагрузках, доступности и простоях. Подход к моделированию основан на моделях строительных блоков, в которых для создания окончательного представления системы используются модели меньшего размера. Первоначально строительные блоки и модель системы представлены для одного серверного узла. Далее описывается подход к моделированию наборов реплик (первичных и запасных узлов).

Все переходы используют экспоненциальное распределение и семантику одного сервера, за исключением случаев, когда это указано. Кроме того, мы принимаем следующие операторы: P{exp} оценивает вероятность внутреннего выражения (exp); E{exp} представляет среднее значение для внутреннего выражения (exp); и #p обозначает количество токенов на месте p.

2.3.1. Простая компонентная модель

На рис. 2.1 показана простая компонентная модель [2.23], которая обычно используется для представления рабочего состояния компонента системы. Положение “On” обозначает рабочее состояние, а положение “Off” представляет состояние отказа. Переходы к болезни и ремонту представляют собой, соответственно, отказ и техническое обслуживание компонента. Переход “Отказ” принимает среднее время до отказа (MTTF) и “Восстановление” учитывает среднее время на ремонт (MTTR).

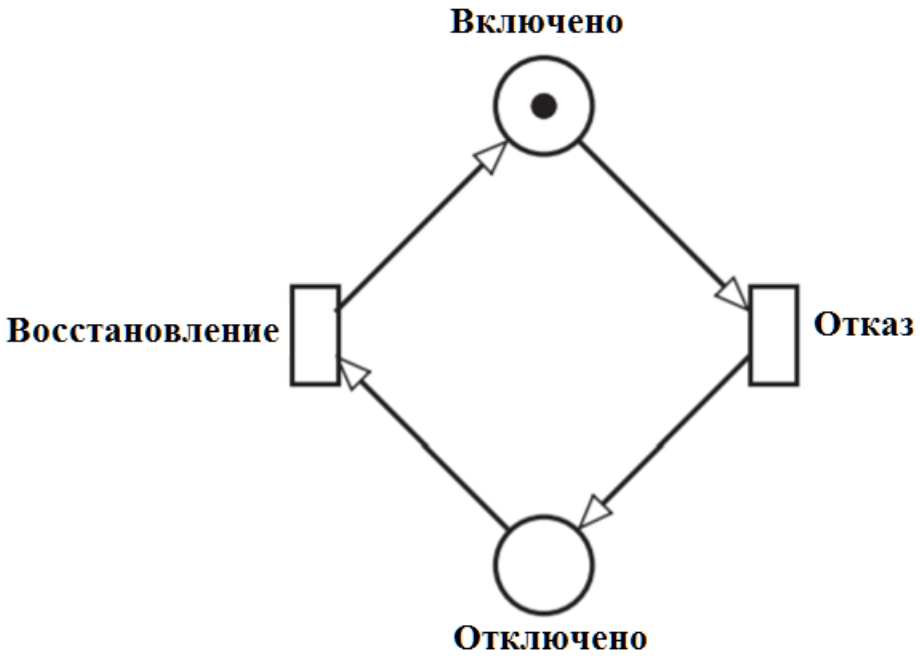


Рис. 2.1. Простая компонентная модель

MTTF - это среднее значение, которое учитывает все компоненты сервера, склонные к отказу, и его можно получить, например, используя блок-схемы надежности (RBD) [2.24]. Среднее время ремонта определяется разработчиком системы и обычно зависит от квалификации обслуживающего персонала и доступности запасных компонентов.

2.3.2. Модель системы

На рис. 2.2 показана модель системы для одного узла базы данных,

которая состоит из 4 блоков: рабочая нагрузка, отклик узла, доступность узла и контроллер.

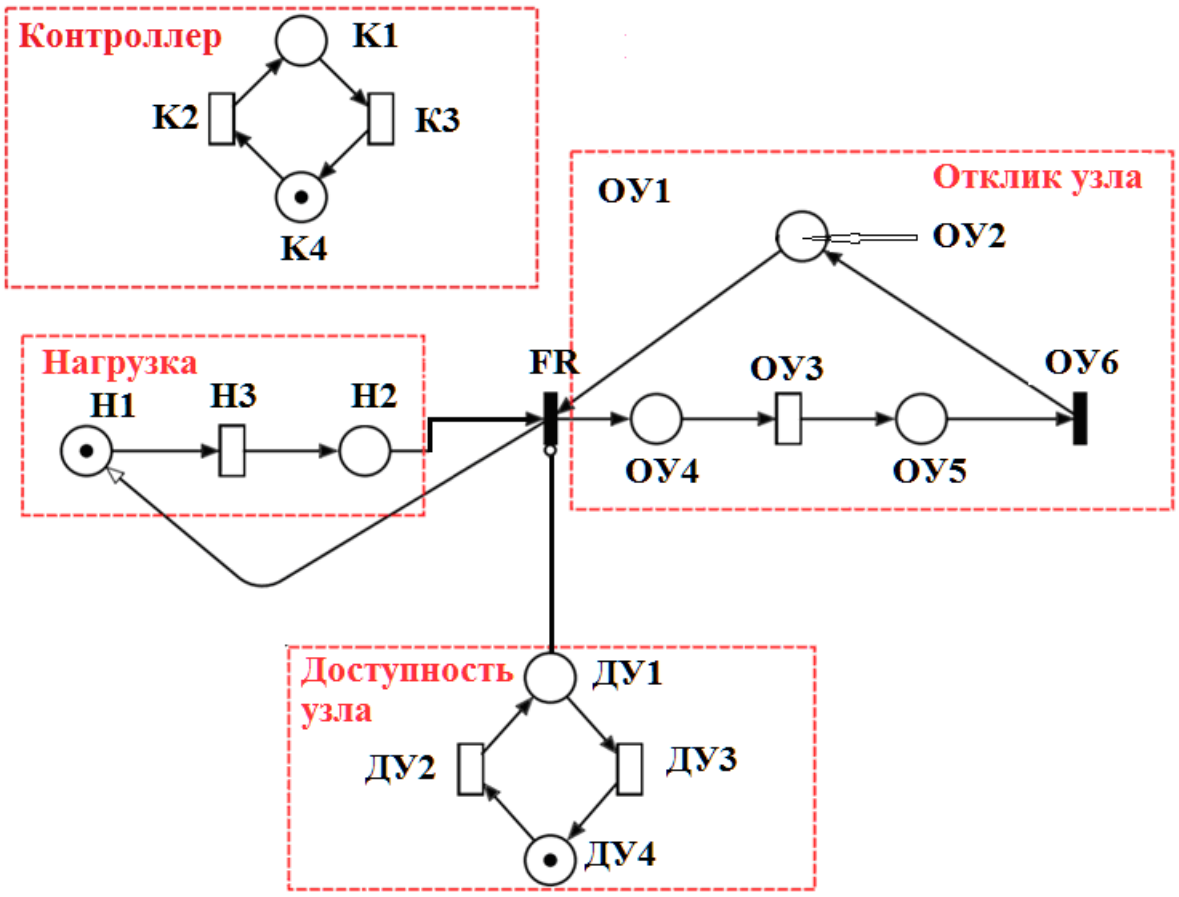


Рис. 2.2. Модель производительности для архитектуры с одним узлом: К1 – ControllerOff; К2 – ControllerFailure; К3 – ControllerRepair; К4 – ControllerOn; ОУ1 – ConnectionPool; ОУ2 – Connections; ОУ3 – ProcessingRequest; ОУ4 – ReceivingRequest; ОУ5 – PreparingResponce; ОУ6 – SendingResponce; FR – ForwardRequest; Н1 – ClientApplication; Н2 – ConnectServer; Н3 – WorkLoadForwarding; ДУ1 – NodeOff; ДУ2 – NodeFailure; ДУ3 – NodeRepair; ДУ4 - NodeOn

Блок рабочей нагрузки обозначает периодическое поступление клиентских запросов, в котором переход от загрузки к загрузке представляет время между запросами. Доступность узла - это простая компонентная модель, которая определяет доступность узла. Таким образом, серверная машина может принять только запрос (переход

ForwardRequest), если сервер находится в рабочем состоянии (метка NodeOn).

Отклик узла - это блок, отвечающий за обработку запросов клиентов. Предполагая, что узел работает, машина может принять новый запрос, если на сервере имеется доступная база данных соединения в его пуле соединений (ConnectionPool). Время обработки запроса и отправки ответа представлено параметром transition ProcessingRequest, который использует семантику бесконечного сервера. После обработки запроса использованное соединение возвращается в пул соединений.

Блок контроллера использует простую компонентную модель для представления облачного контроллера, который отвечает за координацию узлов системы. Поскольку контроллер также подвержен сбоям, он также учитывается в доступности системы. Выражение защиты принимается при переходе ForwardRequest, в котором, в дополнение к доступности узла, также проверяется доступность контроллера: ControllerOn > 0.

На рис. 2.2 показан только один узел, и, таким образом, пропускная способность системы равна пропускной способности узла:

$$TP_{System} = TP_{node} = E\{\#ReceivingRequest\}^* (1/processingDelay), \quad (2.1)$$

в котором processingDelay - это время, связанное с переходом по запросу.

Для обеспечения доступности системы требуется один контроллер и один узел базы данных:

$$A_{System} = P\{(\#NodeOn = 1) \& (\#ControllerOn = 1)\}. \quad (2.2)$$

Время простоя - еще один важный показатель, который рассчитывается с использованием недоступности системы. Предполагая, что один год, время простоя (в часах) составляет:

$$DT_{System} = (1 - A_{System})^*8760. \quad (2.3)$$

2.3.3. Модель системы с набором реплик

Учитывая набор реплик, используются те же строительные блоки,

что и на рис. 2.2, и NodeResponse и NodeAvailability добавляются для каждого дополнительного узла. Дополнительные дуги ингибитора считаются активными запасными узлами только в том случае, если основной выходит из строя.

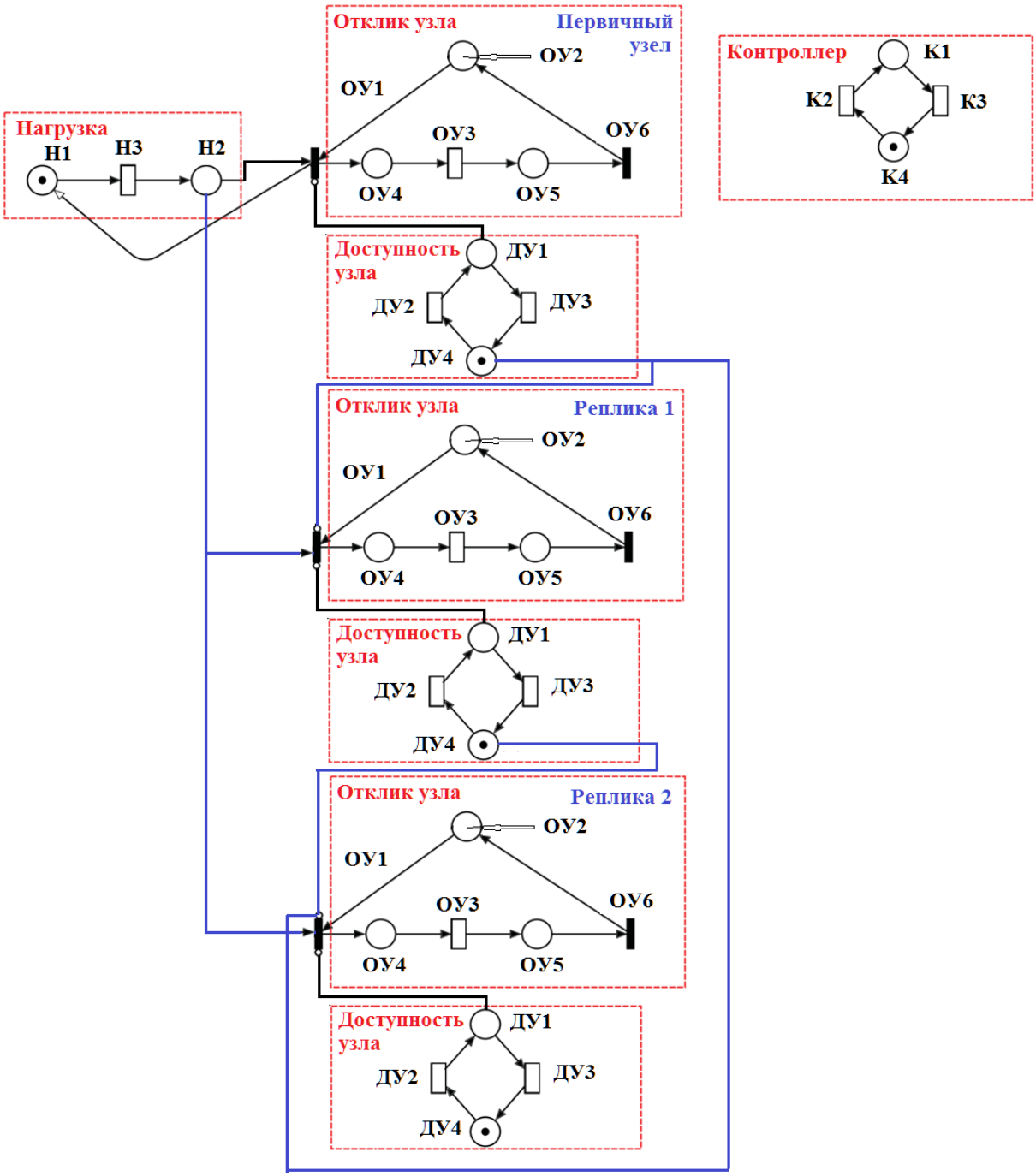


Рис. 2.3. Модель производительности для набора реплик

На рис. 2.3 показана система с основным узлом и двумя запасными

узлами. Добавляется дуга от сервера подключения места к каждому месту ForwardRequest_Rx, и дуги-ингибиторы включаются от узла NodeOn (основного узла) к каждому переходу ForwardRequest_Rx. Кроме того, выражения защиты проверяют доступность контроллера в каждом переходе ForwardRequest_Rx. Дуги ингибитора деактивируют запасные узлы всякий раз, когда основной узел работает.

Пропускная способность системы - это сумма пропускной способности всех узлов, и для модели на рис. 2.3 метрикой является:

TPSystem = (E{#ReceivingRequest} + E{#ReceivingRequest_R1} + E{#ReceivingRequest_R2}) × (1/ processingDelay). (2.4)

Доступность оценивается с учетом контроллера и, по крайней мере, одного рабочего узла СУБД:

A_{System} = P {((#NodeOn = 1 ИЛИ #NodeOn_R1=1 ИЛИ #NodeOn_R2 = 1) И (#ControllerOn = 1)}. (2.5)

2.4. Результаты экспериментов

В этом разделе представлены экспериментальные результаты, демонстрирующие практическую осуществимость предложенной методики моделирования. Первоначально представлена проверка модели, а затем проводятся эксперименты с использованием схемы экспериментов (DoE) [2.25].

2.4.1. Архитектура системы верификации предложенной модели

Для проверки предложенной модели мы приняли частную облачную среду на основе платформы Openstack [2.26] с двумя физическими машинами: одним контроллером и одним вычислительным узлом. Вычислительный узел содержит виртуальную машину с MongoDB СУБД, представляющая подсистему хранения. Кроме того, внешняя машина используется для представления запросов клиентов, которые генерируются

с помощью Yahoo! Эталонный показатель облачного обслуживания [27]. В табл. 2.1 приведены технические характеристики каждой машины.

Таблица 2.1

Облачные машины

Компонент	Узел контроллера	Вычислительный узел	ВМ	Клиент
CPU Процессор	Intel i3-2100	Intel i7-3770	1 core	Intel i5-4200U
ОП	8GB DDR3	8GB DDR3	512MB	8GB DDR3
Жёсткий диск	350GB	250GB	10GB	SSD 250GB
ОС	Centos 7.0	Centos 7.0	Ubuntu 16.04	Centos 7.0

Рабочая нагрузка состоит из 50% операций чтения и 50% операций обновления. Чтобы оценить задержку обработки и интервал времени между прибытиями клиентов, в виртуальной машине MongoDB было получено 30 образцов, выполненных 100 000 операций и с использованием разного количества одновременных потоков (для имитации одновременных пользователей). В табл. 2.2 приведены средние значения нагрузки.

Переход WorkloadForwarding принимает значения, указанные в столбце «прибытие и переход клиента». ProcessingRequest использует значения в столбце «задержка обработки». Учитывая эти значения, была создана модель SPN для каждого прибытия клиента и задержки обработки, была оценена пропускная способность (операции в секунду - операции/с), а затем сопоставлена с реальной системой. В табл. 2.3 представлены результаты для модели и принятого частного облака (sys). Поскольку оцененная пропускная способность с использованием моделей GSPN содержится в доверительных интервалах, полученных с помощью

измерений (95% C.I., уровень значимости α = 0,05), то, следовательно, нет статистических данных, а значит, модель не представляет реальную систему.

Таблица 2.2

Параметры рабочей нагрузки

Пользователи	Прибытие клиента (мс)	Задержка обработки (мс)
1	1.247	1.16×10^{-1}
5	2.566×10^{-1}	9.93×10^{-2}
10	1.717×10^{-1}	1.03×10^{-1}
20	1.420×10^{-1}	1.13×10^{-1}

Таблица 2.3

Результаты модели для проверки пропускной способности

Прибытие клиента (мс)	Система (оп/с)	95% C.I. (оп/с)	Модель (оп/с)
1.247	801.64	[780.51; 823.72]	803.54
2.566×10^{-1}	3897.86	[3868.92, 3939.48]	3909.64
1.717×10^{-1}	5823.43	[5717.55, 5824.11]	5823.29
1.420×10^{-1}	7039.27	[6788.86, 7067.13]	7038.84

2.4.2. Разработка экспериментов (DoE)

Для оценки производительности и доступности системы проводятся два крупных эксперимента, и оба основаны на проектировании экспериментов (DoE) с использованием факториального проектирования l^k.

Первый эксперимент, а именно эксперимент по производительности, учитывает 4 фактора (k = 4) с 2 уровнями (l = 2):

1. количество доступных подключений к базе данных (PoolSize) - 5, 20;

2. задержка обработки одной операции (ProcessingDelay) - 1.29×10^{-1} мс, 6.20×10^{-1} мс;

3. среднее время до отказа вычислительного узла (MTTF) - 50 ч, 228,69 ч; и среднее время на ремонт (MTTR) - 0,775 ч, 3 ч.

Прибытие клиента фиксируется на 4×10^{-1} мс, а пропускная способность является показателем, используемым в этом эксперименте.

Размер POOL - это общий параметр СУБД, который может быть установлен системным администратором для повышения производительности за счет использования большего количества системных ресурсов. Различные значения для ProcessingDelay могут быть получены путем увеличения или уменьшения аппаратной емкости серверов и сети связи. Эта задержка связана с переходом ProcessingRequest. MTTF и MTTR связаны с сервером, и их значения могут отличаться из-за улучшенных аппаратных компонентов или квалифицированной команды технического обслуживания.

Второй эксперимент, а именно эксперимент доступности, учитывает следующие факторы (k = 4) и уровни (l = 2):

1. количество серверов (номер сервера) - 1, 3;

2. среднее время до отказа вычислительного узла (MTTF) - 50 ч, 228,69 ч;

3. среднее время до ремонта вычислительного узла (MTTR) - 0,775 ч, 3 ч;

4. количество контроллеров (Номер контроллера) - 1, 2.

В этом эксперименте в качестве интересующих показателей используются доступность и время простоя. Для этого эксперимента время между запросами клиента и задержкой обработки сохраняется фиксированным на 4×10^{-1} мс и $6{,}20\times10^{-1}$ мс соответственно. Кроме того, количество серверов обозначает один сервер или набор реплик.

Для каждой комбинации уровней факторов была создана и оценена модель GSPN с использованием стационарного анализа. Результаты представлены с использованием ранга эффектов и значений, полученных

при каждом лечении. Эффект - это изменение реакции из-за изменения уровня фактора [25]. Ранги упорядочены в порядке убывания с учетом абсолютных значений всех эффектов.

2.4.3. Эксперимент по выполнимости

Используя DoE предыдущего раздела, в табл. 2.4 представлены основные эффекты и взаимодействия второго порядка (например, ProcessingDelay*PoolSize). В этой работе используются взаимодействия второго порядка, поскольку взаимодействия более высокого порядка обычно незначительны [25].

Таблица 2.4

Ранжирование эффектов для оценки пропускной способности

Факторы и взаимодействия	Эффект (операций / с)
PoolSize	8751.5
ProcessingDelay	−7865.7
PoolSize*ProcessingDelay	6880.6
MTTF	550.4
MTTR	−504.5
MTTR*MTTF	314.8
PoolSize*MTTF	124.3
PoolSize*MTTR	−113.9
ProcessingDelay*MTTF	−111.7
ProcessingDelay*MTTR	102.4

PoolSize оказывает наибольшее влияние на производительность, так как количество доступных подключений также определяет количество одновременных запросов в системе. ProcessingDelay также оказывает большое влияние на пропускную способность системы, поскольку представляет собой задержку для обработки запроса. Если рассматривать

более конкретно, то короткое время обработки увеличивает пропускную способность. В связи с важностью PoolSize и ProcessingDelay, их взаимодействие (PoolSize*ProcessingDelay) оказывает наибольшее влияние среди всех взаимодействий. Учитывая принятые уровни, MTTF и MTTR не оказывают существенного влияния на пропускную способность системы. Тем не менее, эти факторы важны для доступности системы.

В табл. 2.5 представлена пропускная способность (по столбцам) для каждого варианта DoE. Наибольшая пропускная способность связана с наибольшим размером пула (20) и наименьшим значением (последние строки), предполагают меньший размер пула и большую задержку обработки. Короткие MTTF и более высокие MTTR также снижают пропускную способность системы.

Таблица 2.5

Результаты для пропускной способности DoE

MTTF (h)	MTTR (h)	PoolSize	ProcessingDelay (ms)	операций / с
228.69	0.775	20	1.29×10^{-1}	24714.26
228.69	3.000	20	1.29×10^{-1}	24476.92
50.00	0.775	20	1.29×10^{-1}	24419.53
228.69	0.775	20	6.20×10^{-1}	23710.37
228.69	3.000	20	6.20×10^{-1}	23482.67
50.00	0.775	20	6.20×10^{-1}	23427.63
50.00	3.000	20	1.29×10^{-1}	23394.37
228.69	0.775	5	1.29×10^{-1}	22807.63
228.69	3.000	5	1.29×10^{-1}	22588.61
50.00	0.775	5	1.29×10^{-1}	22535.65
50.00	3.000	20	6.20×10^{-1}	22444.12
50.00	3.000	5	1.29×10^{-1}	21589.58
228.69	0.775	5	6.20×10^{-1}	7779.84

MTTF (h)	MTTR (h)	PoolSize	ProcessingDelay (ms)	операций / с
228.69	3.000	5	6.20×10^{-1}	7705.13
50.00	0.775	5	6.20×10^{-1}	7687.12
50.00	3.000	5	6.20×10^{-1}	7364.41

Для лучшей визуализации на рис. 2.4 показан график, демонстрирующий влияние размера пула на пропускную способность. В этом случае разработчик системы должен оценить необходимость закупки более качественных компонентов машины, чтобы увеличить количество доступных соединений, а также сократить время обработки запроса.

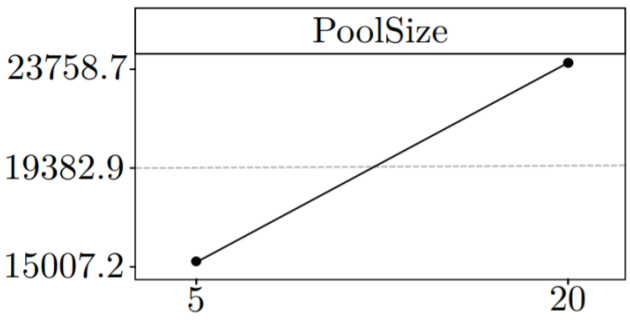


Рис. 2.4. График влияния размера пула на пропускную способность

2.4.4. Эксперимент по обеспечению доступности

В табл. 2.6 и 2.7 представлены результаты по доступности системы.

Таблица 2.6

Ранжирование эффектов для оценки доступности

Факторы и взаимодействия	Эффект (h)
ServerNumber	−183.34
MTTF	−124.04
MTTR	119.13
ServerNumber*MTTF	118.17
ServerNumber*MTTR	−102.57

Факторы и взаимодействия	Эффект (h)
ControllerNumber	−67.27
MTTR*MTTF	−72.18
ControllerNumber*ServerNumber	−0.78
ControllerNumber*MTTF	−0.61
ControllerNumber*MTTR	−0.17

В табл. 2.7 описывается ранг эффектов с использованием времени простоя (в часах в год), поскольку она обеспечивает лучшую визуализацию для небольших значений, связанных с изменением доступности. Количество реплик базы данных (номер сервера), среднее время до сбоя (MTTF) и среднее время ремонта (MTTR) являются важными факторами, влияющими на доступность. Более конкретно, эти факторы ответственны за более чем 100 часов простоя в год, а номер сервера может генерировать почти 200 часов простой. Взаимодействие номера сервера с MTTR и MTTF также оказывает большое влияние на доступность системы. Облачный контроллер также влияет на время простоя, но незначительно по сравнению с другими факторами. Разработчик системы не должен пренебрегать избыточностью узлов СУБД, так как простои могут привести к штрафам из-за нарушения соглашения о качестве обслуживания, потери пользователей и даже невозможности предоставления услуги.

В табл. 2.7 представлены результаты каждого эксперимента. Наибольшая доступность системы связана с 3 серверами (один основной и два запасных сервера), а самые низкие значения сильно связаны с одним сервером (без избыточности).

Таблица 2.7

Результаты по доступности DoE

MTTF (h)	MTTR (h)	ServerNumber	ControllerNumber	Доступность
228.69	0.775	3	2	0.999560
50.00	0.775	3	2	0.999550
228.69	3.000	3	2	0.998476
50.00	3.000	3	2	0.997071
228.69	0.775	1	2	0.996271
228.69	0.775	3	1	0.991902
50.00	0.775	3	1	0.991892
228.69	3.000	3	1	0.990532
50.00	3.000	3	1	0.989264
228.69	0.775	1	1	0.988573
228.69	3.000	1	2	0.986771
50.00	0.775	1	2	0.984395
228.69	3.000	1	1	0.979079
50.00	0.775	1	1	0.976783
50.00	3.000	1	2	0.943137
50.00	3.000	1	1	0.935777

На рис. 2.5 показан график зависимости доступности от числа серверов, который указывает, что время простоя может варьироваться от 229 до 45 часов.

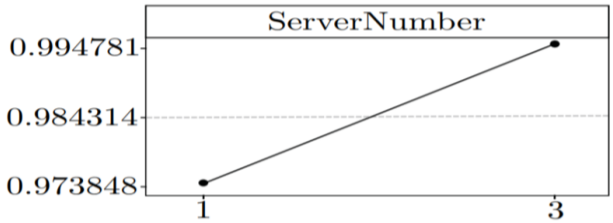


Рис. 2.5. График зависимости доступности от числа серверов

Набор реплик является важной функцией для облачных сред с СУБД NoSQL, поскольку доступность значительно улучшена. Кроме того, предлагаемые модели являются важным инструментом для оказания помощи разработчикам в оценке облаков на основе NoSQL, поскольку репрезентативные системные факторы могут быть оценены без необходимости создания или прерывания работы системы для экспериментов.

2.5. Алгоритм распределения ресурсов в распределенных системах на основе двухкритериальной оценки

Предлагаемый алгоритм будет использовать в основном многокритериальное принятие решений (MCDM) для достижения оптимальной матрицы альтернатив. И это приведет к определению приоритетов выполнения каждого процесса. Чтобы смоделировать эту проблему, мы предполагаем, что набор процессов, обозначаемых символом P, требует ресурсов и ожидает в очереди.

2.5.1. Разработанная модель распределения

Предположим, что у нас есть этот набор из n+1 (n >0) номера процесса, который запрашивает ресурс для использования в распределенной системе, состоящей из нескольких (виртуальных машин). Пусть P описывает набор процессов:

$$P = \{ p_0, p_1, ..., p_n \}. \quad (2.6)$$

Каждая виртуальная машина в системе имеет свою собственную емкость. С другой стороны, процессоры имеют свои собственные атрибуты. Основная идея состоит в том, чтобы выделить и назначить правильный ресурс, который удовлетворял бы всем процессам в системе.

Для достижения этой цели в предложенном алгоритме учтены и проанализированы все атрибуты самого процесса.

В этой модели, принимая два наиболее важных параметра процесса, которые влияют на систему, один - это время выполнения, которое обозначается t_i $(i = 0,...,n)$, а второй будет приоритетом процесса, который рассматривается здесь как время прибытия в очередь ожидания или время начала запроса ресурса или даже время выполнения. Присвоение важности процессам, которые могут быть назначены пользователем или самой системой, таким образом, дает приоритет заданию, которое запрашивает раньше, и это, в свою очередь, приводит к уменьшению времени ожидания для всего процесса, что повышает уровень производительности системы, будем обозначать как ar_i $(i = 0,...,n)$.

Мы предполагаем, что у нас есть распределенная система, состоящая из множества виртуальных машин, содержащих ресурсы, и массив процессов, которые запрашивают для обслуживания доступные ресурсы, а также эти ресурсы ограничены, должны отдавать приоритет одному процессу перед другим. Используя MCDM, метод распределит ресурсы для процесса.

Во-первых, когда процесс p_i просто запрашивает ресурс j в количестве P_{ij}., в этом случае у нас будет матрица процесса, представляющая альтернатив:

$$\tilde{P} = \begin{pmatrix} P_{01} & P_{02} & ... & P_{0m} \\ P_{11} & P_{12} & ... & P_{1m} \\ ... & ... & ... & ... \\ P_{n1} & P_{n2} & ... & P_{nm} \end{pmatrix}. \qquad (2.6)$$

В нашем подходе, зависящем от критериев самих процессов, а не от емкости виртуальных машин, и это при одновременном снижении затрат, достигается основное условие распределения ресурсов:

$$P_{ij} \geq A_{ij}, \tag{2.7}$$

$$A_j = \sum_{i=0}^{n} A_{ij} = T_j - L_j, \tag{2.8}$$

где A_{ij} - объем доступного ресурса j в пуле виртуальных машин для p_i; T_j - общая емкость ресурса j в виртуальных машинах; L_j - объем ресурса j, который уже находится и не свободен на момент запроса.

Предлагаемый алгоритм будет применять как MCDM с использованием PROMETHEE II, и в основном в этом примере он будет использовать только два критерия для принятия решения - это время выполнения процесса (t_i) и порядок запроса или время запроса (ar_i), который, как мы предполагаем, дает этим атрибутам равный приоритет, поэтому этот метод очень эффективен, обеспечивая оптимальное распределение за короткое время. Второй этап нового подхода состоит в том, чтобы проверить, находится ли система в состоянии сохранения из взаимоблокировки, если выполнение задания вызовет взаимоблокировку, то этот процесс будет ждать в очереди. Затем, когда система получит свободные ресурсы, будут обслуживаться процессы, ожидающие в очереди.

В алгоритме мы применили шаги метода PROMETHEE II к распределению ресурсов в распределенных системах, чтобы получить новый эффективный алгоритм. Мы предположили, что существует запрос от многих процессов для виртуальных машин, как показано ниже (табл. 2.8).

Будем использовать два шага метода PROMETHEE II.

Шаг 1: составим все альтернативы в виде нормализованной матрицы в соответствии с прямыми (время выполнения) и косвенными критериями (приоритет запроса). Время выполнения является косвенным, обычно система хочет сначала выполнить самое короткое задание, и процесс,

который запрашивает первым, будет обслуживаться для улучшения QoS всей системы.

Теперь мы применяем нормализацию критериев [2.35]:

- для прямого критерия нормализация такова:

$$t_i^{norm} = \frac{\max\limits_{0\le s\le n}(t_s) - t_i}{\max\limits_{0\le s\le n}(t_s) - \min\limits_{0\le s\le n}(t_s)}; \quad (2.9)$$

- для косвенного критерия нормализация такова

$$ar_i^{norm} = \frac{ar_i - \min\limits_{0\le s\le n}(ar_s)}{\max\limits_{0\le s\le n}(ar_s) - \min\limits_{0\le s\le n}(ar_s)}. \quad (2.10)$$

Таблица 2.8

Процессы, запрашивающие ресурсы

Запрос процесса	Время выполнения (ms)	Запрошенный приоритет
p_0	t_0	ar_0
p_1	t_1	ar_1
p_2	t_2	ar_2
…	…	…
p_n	t_n	ar_n

Шаг 2: находим эволюцию, вычислив разницу по всем альтернативам, затем находим функцию предпочтения для процесса [2.35].

На следующем шаге для каждой пары (p_i, p_j) находим d_1 и d_2:

$$d_1(p_i, p_j) = t_i^{norm} - t_j^{norm}; \quad (2.11)$$

$$d_2(p_i, p_j) = ar_i^{norm} - ar_j^{norm}, \ \forall i, j = 0,1,...,n. \quad (2.12)$$

Для всех критериев (прямого и косвенного) используем функцию предпочтения:

$$Q_1\left(p_i,p_j\right)=f_1\left(d_1\left(p_i,p_j\right)\right); \tag{2.13}$$

$$Q_2\left(p_i,p_j\right)=f_2\left(d_2\left(p_i,p_j\right)\right),\ \forall i,j=0,1,...,n, \tag{2.14}$$

где $Q_1\left(p_i,p_j\right),Q_2\left(p_i,p_j\right)\in[0,1]$.

Алгоритм применяет шесть типов частных функций идентификации предпочтения [2.36]. Средние из них:

1) $f(d)=\begin{cases}0, if\ d\le 0,\\ 1, if\ d>0,\end{cases}$

2) $f(d)=\begin{cases}0, if\ d\le b,\\ 1, if\ d>b,\end{cases}$

3) $f(d)=\begin{cases}0, if\ d\le 0,\\ d/b, if\ 0<d\le b,\\ 1, if\ d>b.\end{cases}$

Последним шагом является получение порядка процессов, которые будут обслуживаться из виртуальных машин, путем вычисления превышения положительных и отрицательных значений:

$$\phi^{+}\left(p_i\right)=\frac{1}{n}\sum_{j=0}^{n}\pi\left(p_i,p_j\right), \tag{2.15}$$

$$\phi^{-}\left(p_i\right)=\frac{1}{n}\sum_{j=0}^{n}\pi\left(p_j,p_i\right), \tag{2.16}$$

где $\pi\left(p_i,p_j\right)=\alpha\cdot Q_1\left(p_i,p_j\right)+\left(1-\alpha\right)\cdot Q_2\left(p_i,p_j\right)$, α – коэффициент важности прямого критерия. Коэффициент α $(0\le\alpha\le 1)$ получен от эксперта.

Далее найдем финальное упорядочение процессов как:

$$W_i=\phi^{+}\left(p_i\right)-\phi^{-}\left(p_i\right),\ i=0,1,...,n. \tag{2.17}$$

Согласно этому методу принятия решений, в зависимости от значений веса (W_i) будет выделен ресурс и определено, какой процесс подходит для обслуживания в это время, чтобы получить более высокую

скорость QoS в распределенных системах. Для выполнения этого решения предложены следующие шаги алгоритма распределения ресурсов, основанного на двухкритериальной оценке процесса.

2.5.2. Алгоритм распределения ресурсов на основе двухкритериальной оценки процесса

Шаг 1: Положить

n – число заданий;

m – число ресурсов;

$\tilde{P}$ – матрица запрашивающих ресурсов (2.6);

$T = (T_1,...,T_m)$ – вектор максимально доступных ресурсов;

$L = (L_1,...,L_m)$ – вектор зарезервированных ресурсов (в начальный момент запроса);

$A = (A_1,...,A_m)$ – вектор доступных ресурсов.

Шаг 2: Пусть q_i – номер p_i в Очереди ожидания; $q = 0$ – длина Очереди ожидания;

Begin

Шаг 3: Применить метод MCMD из PROMETHEE II. Получить вектор весов альтернатив $W = (W_0,...,W_n)$.

Шаг 4: Переупорядочить процессы в невозрастающем порядке, которые мы получаем из метода PROMETHEE II:

$p_{k_0}, p_{k_1}, ..., p_{k_n}$,

где $W_{k_0} \geq W_{k_1} \geq ... \geq W_{k_n}$.

Шаг 5: **For** $h = 0$ **to** ***n***

Begin

Проверить процесс p_{k_h} на отсутствие дедлока с использованием метода «Wait for Graph» или иного любого метода [2.37];

if нет дедлока

then распределить ресурсы для p_{k_h}

else отправить p_{k_h} в Очередь ожидания, $q := q + 1;\ q_{k_h} := q;$

End For

Шаг 6: **until** нет запросов в системе **than** End.

2.5.3. Численный анализ

Пусть для каждого процесса существует два критерия: время выполнения и порядок запроса в системе массового обслуживания (время запроса). Предлагаемый алгоритм будет основываться на этих двух критериях, чтобы найти ранжирование для каждого процесса из попарной матрицы сравнения, которая является результатом значений этих двух критериев с использованием метода MCDM. Итак, с помощью специальной программы (visual PROMETHEE) смоделируем проблему распределения в численном примере (рис. 2.9, 2.10).

PROMETHEE Flow Table

Rank	action	Phi	Phi+	Phi-
1	P0	1,0000	1,0000	0,0000
2	P2	0,2500	0,6250	0,3750
3	P3	0,0000	0,5000	0,5000
4	P1	-0,2500	0,3750	0,6250
5	P4	-1,0000	0,0000	1,0000

Рис. 2.9 Решение о порядке выполнения задач после применения метода MCDM PROMETHEE II

Применив численный пример к программе, которая вычисляет вес процессов W в соответствии с уравнениями PROMETHEE II, получаем таблицу весов (табл. 2.6).

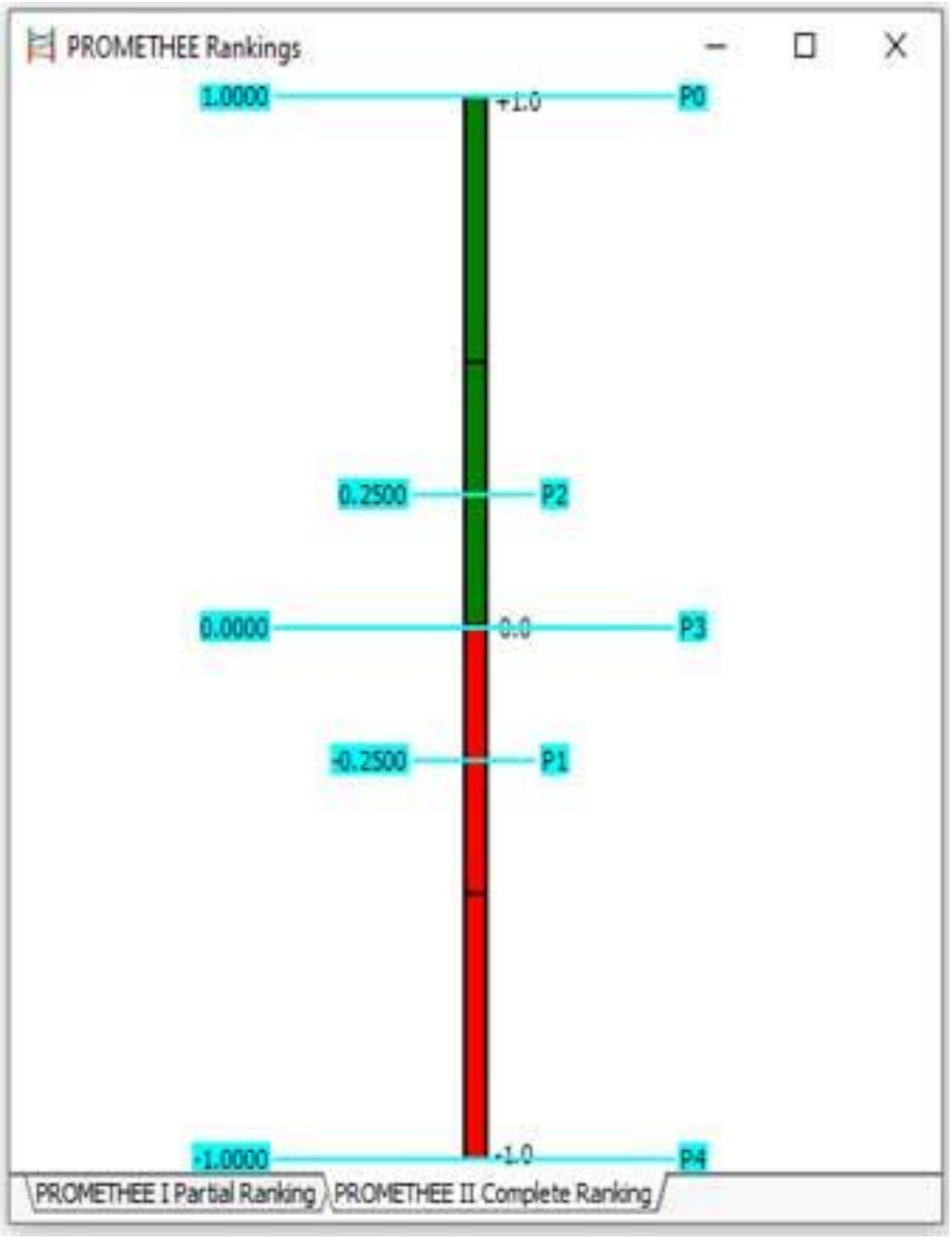


Рис. 2.10. Общий вид ранжирования процессов после нормализации матрицы альтернатив и получения окончательного порядка с весами

В соответствии с весами, порядок процессов будет следующими: p_0, p_2, p_3, p_1, p_4.

Теперь сравним результаты предложенного алгоритма, которые мы получаем из программы моделирования, с традиционным методом, таким

как First Come First Serve (FCFS), как в следующем примере (табл. 2.7).

Таблица 2.6

Вес каждого процесса, определяющий порядок очередности

Процессы	Время выполнения (ms)
p_0	1.00
p_1	-0.25
p_2	0.25
p_3	0.00
p_4	-1.00

Таблица 2.7

Значения характеристик процессов

Запросы процессов	Время выполнения (ms)	Приоритет запроса
p_0	160	500
p_1	300	200
p_2	250	300
p_3	400	400
p_4	700	100

Во-первых, проблему распределения решена с помощью метода FCFS. Этот метод заключается в распределении ресурсов для процессов в зависимости от приоритета запроса. В этом примере порядок процессов в соответствии с этим методом будет следующим: p_0, p_1, p_2, p_3, p_4.

Значения двух критериев для этого порядка процессов показаны на рис. 2.11.

Затем те же процессы были выделены с помощью предложенного алгоритма, который использовал PROMETHEE II, отметим, что в результате использования (MCDM), который зависит от всех критериев в

системе, порядок процессов отличается от первого метода.

Значения всех критериев, полученные в результате применения предложенного алгоритма распределения ресурсов, показаны на рис. 2.12.

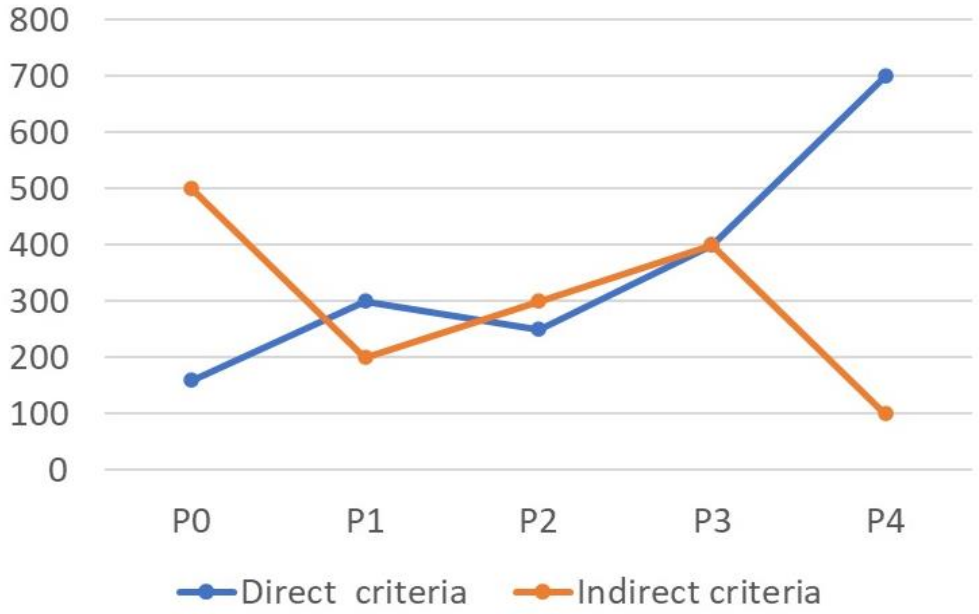


Рис. 2.11. Результат работы метода FCFS, демонстрирующий несбалансированность между двумя критериями при принятии решений о выделении ресурсов

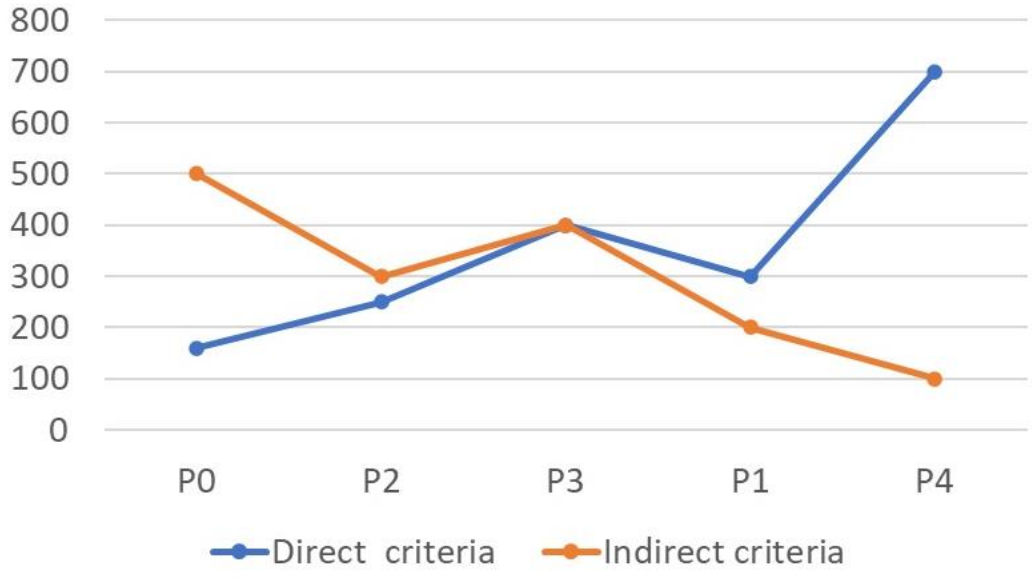


Рис. 2.12. Результат работы предложенного алгоритма, демонстрирующий баланс между двумя критериями при принятии решений о выделении ресурсов

Кроме того, для каждого процесса существует два критерия, поэтому для получения оптимального распределения необходимо учитывать два критерия, чтобы алгоритм процесса находил наилучший баланс между ними, кроме того, предлагаемый алгоритм сокращает время ожидания каждого процесса, пытаясь сначала выполнить процесс короткого времени из очереди с учетом других критериев

Итак, результатом этих задач является распределение ресурсов, сравнивая этот метод и любой другой метод, например FCFS, видно, что этот метод более эффективен в зависимости от более чем одного критерия. Кроме того, найден баланс между важностью критериев. И алгоритм более эффективен в случае многих процессов.

2.6. Выводы

1. На протяжении многих лет многие службы облачных вычислений использовали СУБД NoSQL, но возникают дополнительные проблемы при оценке показателей качества обслуживания (QoS), например, из-за различных рабочих нагрузок и механизмов избыточности.

В работе представлен подход, основанный на GSPN, для оценки производительности облачных вычислительных сред, использующих СУБД NoSQL в качестве системы хранения. Были проведены эксперименты, чтобы продемонстрировать осуществимость предложенной методики. В ходе будущих работ планируется оценить влияние производительности системы на потребление энергии.

2. Предлагаемый алгоритм на основе MCDM находит балансировку системы, в то время как ресурсы уже предоставлены в распределенной системе путем решения стандартной задачи. Одновременно он поддерживает систему в более высоких значениях производительности и QoS с использованием новой стратегии, которая соответствует критериям

задач и выполняется методом MCDM. Результатом стало эффективное распределение ресурсов между задачами, а алгоритм дает системе гибкость при использовании PROMETHEE II для изменения сценариев распределения, придавая некоторым задачам приоритет больше, чем другим, вводя эффективные критерии, которые определяют, как необходимо расширять систему, чтобы использовать критерии ресурсов и задач и комбинировать их для достижения наиболее эффективного распределения ресурсов.

Список источников к главе 2

2.1. Gudivada V.N., Rao D., Raghavan V.V. NoSQL systems for big data management// 2014 IEEE World Congress on Services, 2014.

2.2. Cattell R. Scalable SQL and noSQL data stores// SIGMOD Rec., vol. 39, no. 4, pp. 12-27, 2011.

2.3. Deka G.C. A survey of cloud database systems// IT Professional, vol. 16, no. 2, pp. 50-57, 2014.

2.4. Chalkiadaki M., Magoutis K. Managing service performance in noSQL distributed storage systems// 7th Workshop on Middleware for Next Generation Internet Computing, ser. MW4NG ’12. ACM, 2012.

2.5. Ventura L., Antunes N. Experimental assessment of noSQL databases dependability// 2016 12th European Dependable Computing Conference (EDCC), 2016.

2.6. Abramova V., Bernardino J. NoSQL databases: MongoDB vs Cassandra// International C* Conference on Computer Science and Software Engineering, ser. C3S2E ’13. ACM, 2013, pp. 14-22.

2.7. Osman R., Knottenbelt W.J. Database system performance evaluation models: A survey// Performance Evaluation, vol. 69, no. 10, pp. 471 - 493, 2012.

2.8. Dede E. et al. Performance evaluation of a mongoDB and hadoop platform for scientific data analysis// 4th ACM Workshop on Scientific Cloud Computing, ser. Science Cloud ’13. ACM, 2013.

2.9. Klein J. et al. Performance evaluation of noSQL databases: A case study// 1st Workshop on Performance Analysis of Big Data Systems, ser. PABS ’15. ACM, 2015.

2.10. Tang E., Fan Y. Performance comparison between five noSQL databases// 2016 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 105-109, 2016.

2.11. Gomes C., Tavares E.A.G., de Nogueira M.O. Jr. Energy consumption evaluation of noSQL DBmss// 15 WPerformance - Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2016.

2.12. Bruneo D. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems// IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 560-569, 2014.

2.13. de Lima Q.V. et al. Performability evaluation of emergency call center// Performance Evaluation, vol. 80, pp. 27 - 42, 2014.

2.14. Kirsal Y. et al. Analytical modeling and performability analysis for cloud computing using queuing system// in 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015.

2.15. Ghosh R. et al. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach// 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, 2010.

2.16. Gandini A. et al. Performance evaluation of noSQL databases// Computer Performance Engineering. Springer International Publishing, 2014.

2.17. Dipietro S., Casale G., Serazzi G. A queuing network model for performance prediction of Apache Cassandra// 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), 2017.

2.18. Osman R., Piazzolla P. Modeling replication in noSQL datastores// Quantitative Evaluation of Systems. Springer International Publishing, 2014.

2.19. Davoudian A., Chen L., Liu M. A survey on noSQL stores// ACM Comput. Surv., vol. 51, no. 2, pp. 40:1-40:43, 2018.

2.20. Murata T. Petri nets: Properties, analysis and applications// Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, 1989.

2.21. Marsan M. et al. An introduction to generalized stochastic Petri

Nets// Microelectronics Reliability, vol. 31, no. 4, pp. 699 - 725, 1991.

2.22. Balbo G. Introduction to Stochastic Petri Nets. Springer Berlin Heidelberg, pp. 84 - 155, 2001.

2.23. Silva B. et al. Astro: An integrated environment for dependability and sustainability evaluation// Sustainable Computing: Informatics and Systems, vol. 3, no. 1, pp. 1 - 17, 2013.

2.24. Trivedi K.S., Hunter S., Garg S., Fricks R. Reliability analysis techniques explored through a communication network example. North Carolina State University. Series/Report No.: TR-96/32. 1996.

2.25. Montgomery D. Design and Analysis of Experiments. John Wiley & Sons, 2008.

2.26. Openstack cloud software. https://www.openstack.org/software/, 2018.

2.27. Cooper B.F. Yahoo! cloud system benchmark (YCSB). https://github.com/brianfrankcooper/YCSB, 2018.

2.28. Thamsen L, Verbitskiy I, Beilharz J, Renner T, Polze A, Kao O. Ellis: Dynamically Scaling Distributed Dataflows to Meet Runtime Targets. Proc. Int/ Conf. Cloud Comput Technol Sci CloudCom. 2017;(37)146–153.

2.29. Silberschatz A., P. Baer Galvin, Gagne G. Operating Systems Concepts. New York: John_Wiley_&_Sons. 2008;(8):741-750.

2.30. Thamsen L., Renner T, Kao O. Continuously Improving the Resource Utilization of Iterative Parallel Dataflows. Proceedings of the 6th International Workshop on Big Data and Cloud Performance, ser. DCPerf 2016. IEEE. 2016;1(4):1–6.

2.31. Castillo G., Rouskas N., Harfoush K. Efficient QoS resource management for heterogeneous Grids. 22nd. IEEE International Parallel and Distributed Processing Symposium (IPDPS'08), Miami, Florida, US. 2008;1-15.

2.32. Jiang H., Ni T. PB-FCFS-a task scheduling algorithm based on FCFS and backfilling strategy for grid computing. Proceedings of Joint

Conferences on Pervasive Computing (JCPC). 2009; 507- 510.

2.33. Triantaphyllou E. Multi-Criteria Decision-Making Methods in Multi-Criteria Decision-Making Methods, a Comparative Study. Applied Optimization. 2000; 44(1): 5–21.

2.34. Chen L., Xu Z., Wang H., Liu S. An ordered clustering algorithm based on K-means and the PROMETHEE method. International Journal of Machine Learning and Cybernetics. 2018; 9(6):917-926.

2.35. Chakraborty S., Yeh C. H. A Simulation Based Comparative Study of Normalization Procedures in Maldistributed Decision Making. Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases.2007; 102–109.

2.36. Mareschal B., Brans Jean-Pierre. PROMETHEE methods. International Series in Operations Research and Management Science.2014;78(2):163-195.

2.37. Yu L., Chen L., Cai Z., Shen H., Liang Y., Pan Y. Stochastic Load Balancing for Virtual Resource Management in Datacenters. IEEE Transactions on Cloud Computing. 2018; 8(2):459–472.

3. Проектирование самонастраиваемой репликации СУБД на основе форсированного обучения

3.1. Реплицируемые облачные СУБД и настраиваемые критерии

Распределенные системы, а именно масштабируемые облачные системы управления базами данных (СУБД), охватывают все большее число настраиваемых критериев, которые могут серьезно повлиять на производительность системы [3.5]. Это особенно верно для реплицируемых СУБД, поскольку возможности репликации часто тесно связаны с общей конструкцией системы, и отсутствие надлежащей настройки может повлиять на надежность системы, ухудшая качество обслуживания. Кроме того, внутренние характеристики каждой рабочей нагрузки также напрямую влияют на то, как работает механизм СУБД.

Количество и тип настраиваемых критериев, доступных в каждой СУБД [3.4, 3.16, 3.18], различаются, но в целом они позволяют настроить общий набор свойств, таких как доступное пространство памяти, количество разрешенных параллельных рабочих мест, а также размер списка выполнения для конкретных задач. Что касается репликации, обычно системы позволяют настроить тип рассматриваемого механизма репликации, количество активных экземпляров, допустимые задержки подтверждения и то, как далеко может дрейфовать данная реплика по согласованности данных, прежде чем надежность системы будет поставлена под сомнения.

Воспринимаемая производительность системы обычно измеряется через количество выполненных операций за определенный промежуток времени, которое часто определяется как экзогенное свойство, предоставляемое внешними инструментами бенчмаркинга, а не как внутреннее свойство системы. В итоге, достижение наилучшей возможной конфигурации является результатом многоступенчатого процесса, в

котором важную роль играют методы проб и ошибок, возлагающие ответственность за настройку на администратора базы данных (DBA) [3.5].

Быстрое расширение методов автономного и машинного обучения в настоящее время наталкивает на рассмотрение оптимизации систем этих подходов таких аспектов, как обнаружение и распознавание образов, а также самонастройки систем.

Классические подходы контролируемого обучения разделяют на отдельные этапы: период строго для обучения и второй - период прогнозирования. Поскольку этапы не связаны между собой, как правило, включение новых шаблонов данных в модель подразумевает новый период обучения. Также используют методы, которые объединяют эти два этапа, например, обучение с подкреплением (RL) [3.20].

Рассмотрим самонастраивающийся механизм, основанный на обучении с подкреплением, для конфигурирования и регулирования в режиме реального времени гибридного промежуточного программного обеспечения репликации в паре с системой СУБД. Когда запросы поступают в промежуточное программное обеспечение репликации, они разбиваются на сегменты, т. е. набор логических разделов, основанных на рабочей нагрузке, которые затем назначаются отдельным репликам СУБД. Система способна исследовать конфигурацию промежуточного программного обеспечения и настроить ее на идеальную конфигурацию в режиме реального времени, не требуя затрат на перезапуск промежуточного программного обеспечения или базовой СУБД. Более того, она делает это динамично, конфигурация постоянно корректируется, чтобы отразить наилучшие значения для текущей рабочей нагрузки.

3.2. Проектирование системы

На основе описанной выше концепции, предложена архитектура, изображенная на рис. 3.1.

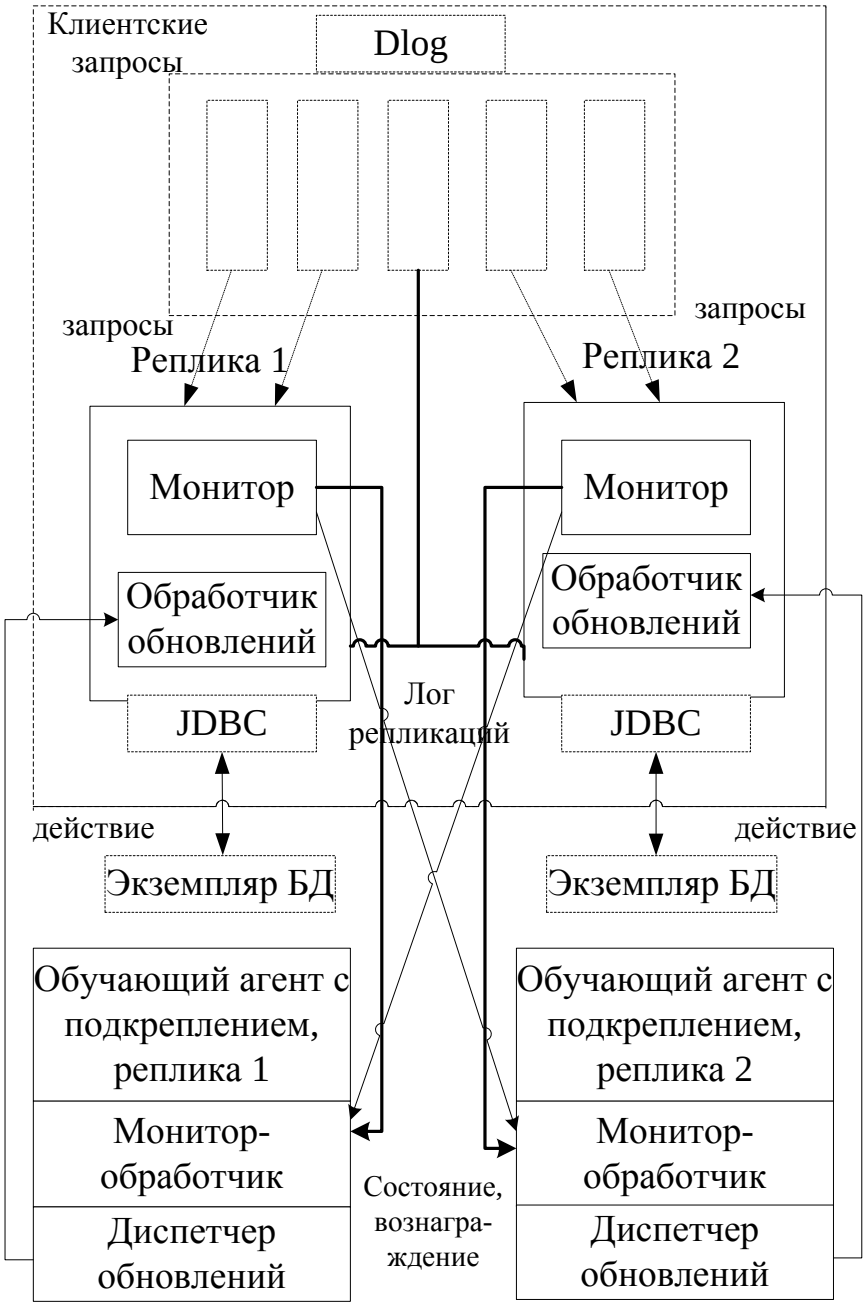


Рис. 3.1. Системная архитектура

3.2.1. Общее описание

Система состоит из трех отдельных компонентов, а именно:

1. узлы реплики промежуточного программного обеспечения, содержащие монитор метрик и обработчик обновлений;

2. система обучения с подкреплением, содержащая обработчик монитора метрик, диспетчер обновлений и сам агент обучения с подкреплением;

3. базовая СУБД. Промежуточное программное обеспечение репликации построено на основе Apache Bookkeeper Distributed Log (DL).

Промежуточное программное обеспечение репликации получает входящие запросы JDBC от клиентов, которые затем передаются в группу распределенных экземпляров лога. Распределенный лог - это эффективное и высокопроизводительное решение для обеспечения репликации и обмена репликами. Его интерфейс учитывает две роли в своей архитектуре, а именно: авторы и читатели. Запросы проходят процедуру сегментирования [3.8], разделяя их на слоты в соответствии с настраиваемым числом экземпляров базы данных (БД) и сопоставляя каждый экземпляр распределенного лога группе сегментов. Узлы реплики выполняют назначенные им клиентские запросы, а также записывают результирующие изменения состояния в логах репликации. Они также получают реплицированные изменения состояния от других реплик и соответствующим образом обновляют свои записи. Промежуточное программное обеспечение репликации основано на протоколе активной репликации [3.24] с полуактивным вариантом [3.13]. С одной стороны, когда сегменты назначаются реплике, эта реплика становится основной, обрабатывая запросы для этого сегмента. С другой стороны, одна и та же реплика действует как резервная копия, включая обновления других первичных реплик, отвечающих за другие сегменты.

Система обучения с подкреплением построена из набора подкомпонентов: монитора и обработчика обновлений, которые действуют как датчик в каждой реплике, а также обработчика монитора, диспетчера

обновлений и агента обучения. Архитектура содержит экземпляр для каждого из этих компонентов на реплику. Монитор проверяет и агрегирует метрики о каждой реплике, которые затем передаются обработчику монитора. Затем результаты передаются в агент обучения с подкреплением, который настраивает базовую конфигурацию и инструктирует реплики по изменению своих конфигураций с помощью диспетчера обновлений и датчика локального обработчика обновлений реплики. Изменения применяются в режиме онлайн, без необходимости перезапуска промежуточного программного обеспечения или базовых экземпляров БД.

Были рассмотрены все переменные для настройки, доступные с помощью промежуточного программного обеспечения, как показано в табл. 3.1.

Таблица 3.1

Регулируемые конфигурации, рассматриваемые для настройки в режиме онлайн

Конфигурация	Описание
db.pool	Размер пула подключений
dlog.reader.threads	Количество рабочих потоков
db.writeset.transactions	Максимальный размер пакета
db.writeset.delay	Задержка между записями в логах репликации

Набор возможных действий для обучающегося с подкреплением состоит из постепенных изменений этих переменных. Кроме того, следует отметить, что каждая реплика имеет независимый агент обучения с подкреплением, управляющий ее переменными настройки, которые могут быть развернуты на другой машине. Это позволяет добавлять новые реплики в кластер без необходимости переобучать модель для уже существующих реплик. Кроме того, поскольку разные реплики

обрабатывают разные сегменты, рабочая нагрузка может варьироваться между ними. Настраивая каждую из них по отдельности, мы получаем конфигурацию, оптимизированную для рабочей нагрузки каждой реплики.

3.2.2. Компоненты

Чтобы обеспечить механизм обучения с подкреплением, проектирование включает в себя набор пользовательских компонентов для наблюдения и сбора метрик, а также для инициирования необходимых изменений конфигурации. Компоненты разделены на две группы: те, которые относятся к реплике, датчикам и тем, которые относятся к агенту обучения подкреплению.

Монитор. Компонент монитора развертывается в каждой реплике промежуточного программного обеспечения репликации. Он исследует набор локальных свойств, которые затем передаются в механизм обучения. Недостатком этого компонента является служба очередей публикации и подписки [3.9], которая периодически передает показания в компонент обработчика монитора. На практике этот компонент снабжает алгоритм обучения с подкреплением обновлениями состояния, формирует обновления. Обновления отправляются каждые 15 секунд.

Обработчик обновлений. Компонент обработчика обновлений развертывается в каждой реплике промежуточного программного обеспечения репликации. Он получает асинхронно действия от диспетчера обновлений и применяет их непосредственно в каждом узле промежуточного программного обеспечения, позволяя динамически применять изменение без необходимости перезапуска промежуточного программного обеспечения или системы БД.

Обработчик Монитора. Компонент обработчика монитора развертывается вне узлов реплики, в каждом из Агентов обучения с подкреплением. Он собирает метрики, отправленные агентами

мониторинга в каждой реплике, подписываясь на их обновления через службу публикации и подписки. Эти данные будут использоваться для определения текущего состояния окружающей среды и вознаграждения, связанного с действием.

Диспетчер обновлений. Компонент диспетчера обновлений также развертывается на каждом из агентов обучения с подкреплением. Он рассматривает набор действий, назначенным компонентом RL, и запускает обработчики обновлений для внесения изменений в новые конфигурации.

3.2.3. Агент обучения с подкреплением

Каждая реплика управляется отдельным агентом обучения с подкреплением. Компонент RL отвечает за анализ данных, поступающих с Монитора через Обработчик монитора. На каждом шаге алгоритм RL характеризует входящие необработанные данные из компонента Монитора в состояние, которое будет передано алгоритму RL, которое будет преобразовано политикой агента в действие. Это действие затем применяется к среде, реплике промежуточного программного обеспечения. Настройка переменных конфигурации выполняется динамически. Это означает, что значения конфигурации постоянно корректируются в ответ на изменения рабочей нагрузки.

Пространство поиска, связанное с этими стратегиями, характеризуется комбинаторным набором, построенным из всех возможных состояний и действий. Собранные данные, состояние, характеризуются переменными, указанными в табл. 3.2, набором дискретных переменных, с которыми связано большое пространство поиска. Выбранным алгоритмом был алгоритм глубокого Q-обучения, так как он может вместить большие пространства поиска. Этот механизм включает в себя сложную нейронную сеть поверх алгоритма Q-обучения [3.23]. Учитывая этот выбор, пространство действий было разделено на

подмножество из 10 возможных вариантов, показанных в табл. 3.3.

Таблица 3.2

Набор элементов, составляющих состояние в процессе RL

Конфигурация	**Описание**
db.pool	Текущий размер пула подключений
dlog.reader.threads	Текущее количество рабочих потоков
db.writeset.transactions	Текущий максимальный размер пакета
db.writeset.delay	Текущая задержка между записями в логах репликации
ClientRequests	Выполненные транзакции
Txn Written	Обновления состояния транзакций (Txn), отправляемые в логи репликации
Replicated Txn	Обновления состояния реплицированных транзакций, применяемые к СУБД
ClientRequests	Количество новых запросов клиентов на эту реплику
rep_Txn in DL	Транзакции, отправляемые в логи репликации другими репликами

Таблица 3.3

Действия, рассмотренные в процессе RL

Действия	**Описание**
Без действий	Ничего не изменилось
Увеличение db.pool	Увеличение на 1
Уменьшение db.pool	Уменьшение на 1
Увеличение dlog.reader.threads	Увеличение на 1
Уменьшение dlog.reader.threads	Уменьшение на 1
Увеличение db.writeset.transactions	Увеличение на 100

Действия	**Описание**
Уменьшение db.writeset.transactions	Уменьшение на 100
Установка специальных параметров db.writeset.transactions	Установка на 0
Увеличение db.writeset.delay	Увеличение на 100
Уменьшение db.writeset.delay	Уменьшение на 100

Состояния. Состояния, учитываемые методом RL, представляют собой композицию текущих значений переменных, изображенных в табл. 3.1, которые затем дополняются набором метрик, характеризующих общую производительность системы в каждой реплике. Эти показатели представляют собой среднее значение за период, охватываемый между двумя последовательными показаниями метрик. Полный набор элементов, составляющих состояние данной модели обучения с подкреплением, представлен в табл. 3.2.

Метрики позволяют установить связь между количеством запросов, полученных в данной реплике, и количеством запросов, выполненных в этой же реплике. Кроме того, количество полученных и выполненных транзакций на других репликах системы (которые не будут оптимизированы) также являются частью состояния.

Это позволяет установить соотношение между количеством выполненных реплицированных запросов и общим количеством запросов, которые другие реплики уже отправили в распределенный лог для сохранения. Таким образом, он позволяет системе узнать, является ли более низкая пропускная способность реплицируемых транзакций следствием недостаточной производительности самой реплики или реплики, отправляющей обновления.

Текущие значения корректируемых переменных становятся частью состояния, чтобы нейронная сеть могла устанавливать связи между

значениями переменных и между ними и собранными метриками, поскольку возможные действия, выходные данные нашей нейронной сети, не отражают фактические значения переменных, а скорее постепенные изменения в них.

Действия. Действия, решаемые нейронной сетью, составляют возможные изменения, которые могут быть внесены на каждом шаге в окружающей среде. При использовании глубокого Q-обучения переменные отбирались в виде инкрементных изменений. Возможный набор действий представлен в табл. 3.3. Для предотвращения сбоев системы для каждой переменной настройки был установлен верхний и нижний предел. Значения приращения и уменьшения были выбраны методом проб и ошибок. Цель состояла в том, чтобы выбрать приращения, которые не были бы настолько малы, чтобы они не оказали существенного влияния на производительность системы, но не настолько велики, чтобы количество возможных значений для каждой переменной стало очень низким (с учетом установленных границ). Таким образом, на каждом шаге алгоритма выполняется только одно из вышеупомянутых действий. Это означает, что на каждом шаге изменяется только одна переменная, и ее значение изменяется лишь на небольшую величину, как описано в табл. 3.3.

Переменная конфигурации db.writeset.transactions может быть установлена в специальное значение 0. Когда это значение установлено, ограничение на количество транзакций, которые могут быть записаны в каждом пакете, удаляется.

Функция вознаграждения. Поскольку рассматриваемая среда ограничена репликацией базы данных и общим выполнением базы данных, влияние может быть напрямую связано с общей пропускной способностью. Более высокая пропускная способность обеспечивает лучший результат. Следовательно, функция вознаграждения связана с пропускной способностью и определяется как сумма всех последних

средних показателей, которые относятся к реплике. Поскольку все средние значения находятся в транзакциях в каждом моменте времени, нет необходимости применять к значениям какую-либо нормализацию или преобразование. В этом случае все компоненты вознаграждения имеют одинаковый вес по отношению к вычисленному вознаграждению:

$$reward = ClientRequests + TxnWritten + ReplicatedTxn \quad (3.1)$$

3.2.4. Механизм обучения с подкреплением

Предлагаемый механизм рассматривает RL, основанный на глубоком Q-обучении. В рамках подхода RL Агент отслеживает определенную среду и принимает решение о действии с помощью своей Политики. Цель агента состоит в том, чтобы найти наилучшую возможную Политику, которая максимизирует рассматриваемое вознаграждение. Значения вознаграждения вычисляются после применения Действий к Среде с использованием рассматриваемой Функции вознаграждения. Система изображена на рис. 3.2.

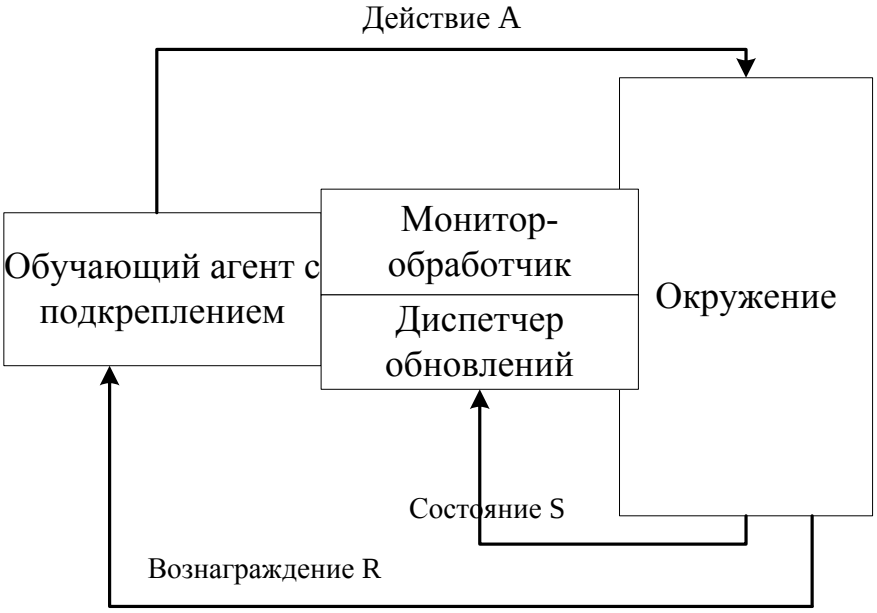


Рис. 3.2. Обучение с подкреплением в окружающей среде

Q-обучение устанавливает таблицу, которая определяет политику

агента. Он сопоставляет возможные действия и состояния, присваивая каждой комбинации значение q. В начале процесса обучения все значения q равны нулю. Политика определяет, что для каждой среды мы должны выбрать действие с наибольшим значением q, соответствующим наибольшему вознаграждению. Однако всегда выбирать действие с наибольшим значением q может помешать обнаружению альтернативных вероятных действий, которые могут ухудшиться по мере увеличения числа состояний. Таким образом, небольшой процент действий выбирается случайным образом.

Введение нейронной сети вместо таблицы состояний позволяет Deep Q-Learning учитывать больше комбинаций наборов состояний. Каждый выходной узел представляет возможное действие, и значение, рассчитанное сетью для этого узла, будет эквивалентным q действия. Рассматриваемая нейронная сеть изображена на рис. 3.3. Он содержит два скрытых слоя по 16 нейронов в каждом. На каждом шаге выбранное действие будет иметь наибольшее значение q. Вознаграждения используются для корректировки веса нейронной сети путем обратного распространения. Согласно рис. 3.3, выбранным действием будет увеличение потоков dlog.reader., потому что на изображенном шаге оно имеет наибольшее значение q.

3.2.5. Механизм предварительной оценки

Оценка системы была построена с учетом эталона TPC-C, разработанного специально для оценки систем баз данных OLTP. Спецификация TPC-C моделирует реальный сценарий, в котором компания, состоящая из нескольких складов и районов, обрабатывает заказы, размещенные клиентами. Рабочая нагрузка определяется по 9 таблицам, управляемым набором транзакций, состоящим из пяти различных транзакций, а именно: Новый заказ, Оплата, Статус заказа, Доставка и Уровень запасов. Каждая транзакция состоит из нескольких

операций чтения и обновления, где 92% составляют операции обновления, что характеризует это как большую нагрузку на запись.

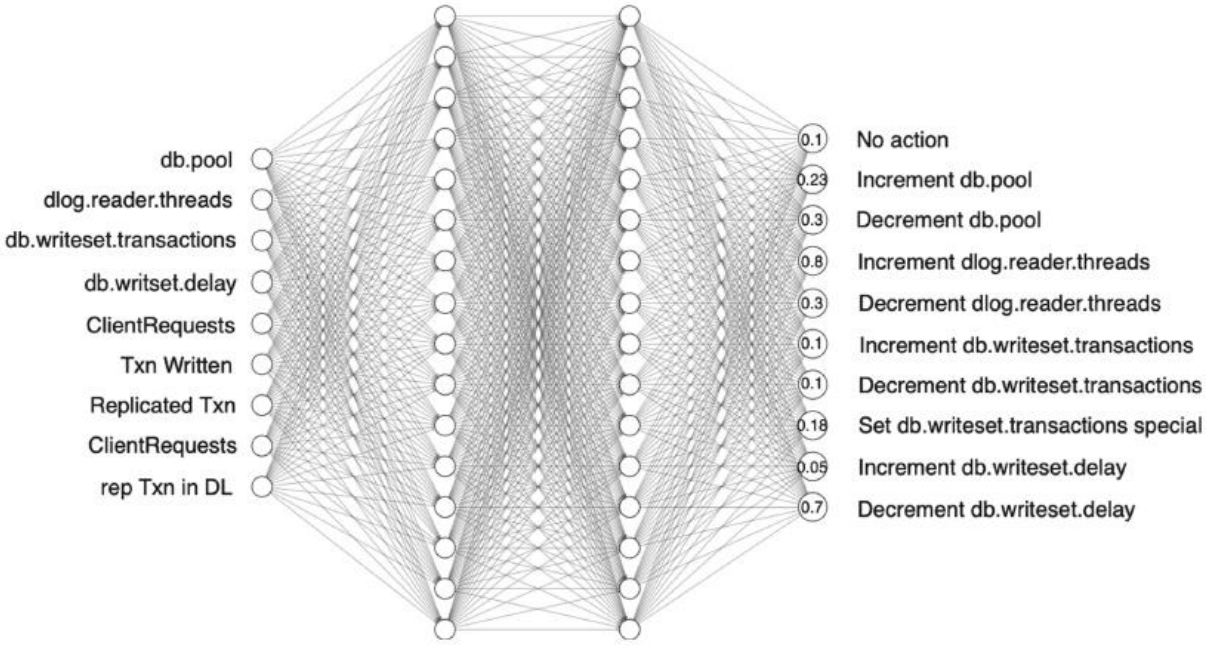


Рис. 3.3. Нейронная сеть, используемая в агентах обучения с подкреплением. Вычисленные значения q изображены справа в паре с соответствующим действием

Таблица 3.4

Конфигурация, настроенная с учетом обучения с подкреплением из базовой конфигурации. Baseline и цикл приведены к транзакциям в секунду (Txn/сек)

Metric	**Baseline**	**RL#1**	**Прирост**	**RL#2**	**Прирост**
ClientRequests-R1	80.80	94.07	+16.42%	91.87	+13.70%
Replicated Txn-R1	35.24	55.56	+57.64%	55.00	+56.05%
Txn Written-R1	27.57	61.73	+123.92%	91.87	+233.25%
ClientRequests-R2	178.20	129.51	−27.32%	157.82	−11.44%
Replicated Txn-R2	27.16	61.75	+127.40%	91.56	+237.18%
Txn Written-R2	31.86	129.51	+306.44%	150.08	+370.99%
Avg Reward-R1 Avg	143.61	211.35	+47.17%	238.74	+66.24%

Metric	**Baseline**	**RL#1**	**Прирост**	**RL#2**	**Прирост**
Reward-R2	237.22	320.78	+35.22%	399.46	+68.39%
Metric	Baseline	RL#4	Прирост	RL#6	Прирост
ClientRequests-R1	80.80	99.96	+23.71%	86.04	+6.49%
Replicated Txn-R1	35.24	52.57	+49.15%	52.09	+47.79%
Txn Written-R1	27.57	99.96	+262.59%	86.04	+212.11%
ClientRequests-R2	178.20	142.30	−20.15%	207.00	+16.16%
Replicated Txn-R2	27.16	99.96	+268.09%	68.21	+151.16%
Txn Written-R2	31.86	142.30	+346.57%	111.55	+250.09%
Avg Reward-R1 Avg Reward-R2	143.61 237.22	252.48 384.55	+75.81% +62.11%	224.17 386.75	+56.09% +63.03%
Metric	Baseline	RL#8	Прирост	RL#10	Прирост
ClientRequests-R1	80.80	111.92	+38.52%	112.95	+39.79%
Replicated Txn-R1	35.24	30.45	−13.59%	69.08	+96.01%
Txn Written-R1	27.57	75.46	+173.74%	112.95	+309.73%
ClientRequests-R2	178.20	220.57	+23.77%	205.47	+15.30%
Replicated Txn-R2	27.16	68.20	+151.15%	98.25	+261.81%
Txn Written-R2	31.86	96.73	+203.57%	94.26	+195.82%
Avg Reward-R1 Avg Reward-R2	143.61 237.22	217.84 385.50	+51.69% +62.51%	294.99 397.99	+105.41% +67.77%

Вычислительный эксперимент. TPC-C был настроен в конфигурации, включающей 150 хранилищ с нагрузкой 50 клиентских соединений с хранилищем. Кроме того, компонент репликации промежуточного программного обеспечения был настроен для размещения 25 хранилищ на один экземпляр распределенного журнала. Тесты проводились на локальном сервере, построенном на базе процессора Ryzen 3700, 16 ГБ оперативной памяти и 2 жёстких накопителей (одним из

которых является NVME), на котором размещены все сервисы. Реплика 1 (R1) использовала драйвер NVME, в то время как реплика 2 (R2) использовала другой SSD-накопитель.

Оценка была достигнута путем запуска теста TPC-C, и, пока он отправлял транзакции для фиксации в базовой базе данных через промежуточное программное обеспечение репликации, агент обучения с подкреплением был развернут на отдельной машине.

В общей сложности было проведено 10 циклов обучения, все они начинались с одной и той же базовой конфигурации. Начальная базовая конфигурация была построена на основе предположений исследователей приемлемой конфигурации для используемой системы. Начальная базовая конфигурация одинакова для всех циклов обучения, так что начальное состояние промежуточного программного обеспечения не отличается между ними.

Первые 5 циклов были скорректированы для более быстрого процесса обучения, в большей степени ориентированного на исследования. Эта серия циклов состояла из 100 шагов, обновляя нагрузку нейронной сети каждые 15 шагов, и с вероятностью случайного выбора действия 20%.

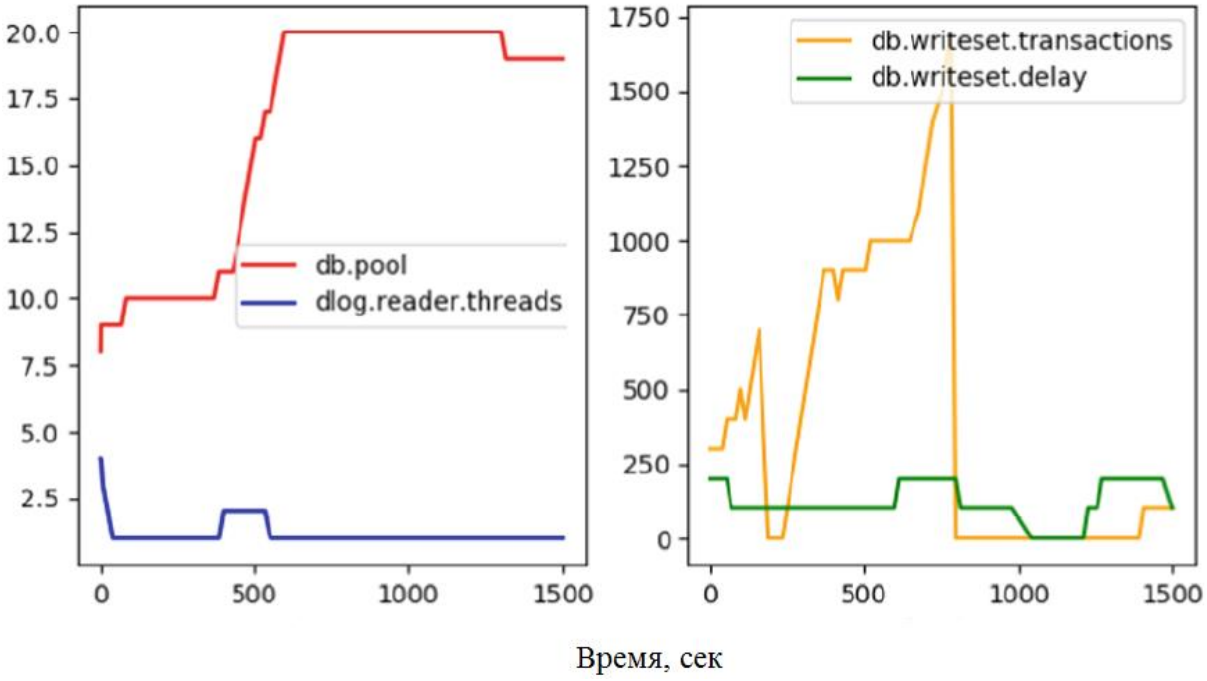
В последних 5 циклах нагрузка обновлялась каждые 20 шагов, и действия выбирались случайным образом только в 10% случаев. Количество шагов также было увеличено до 120. Количество шагов было выбрано таким образом, чтобы агент обучения с подкреплением был активен примерно в течение того же времени, которое требуется для завершения теста. Каждый шаг включает в себя выполнение одного из действий, описанных в табл. 3.3.

Стоит отметить, что механизм обучения превосходит обновления состояния окружающей среды в короткие промежутки времени, что в противном случае привело бы к большим задержкам в процессе обучения, поэтому была внедрена пользовательская система мониторинга.

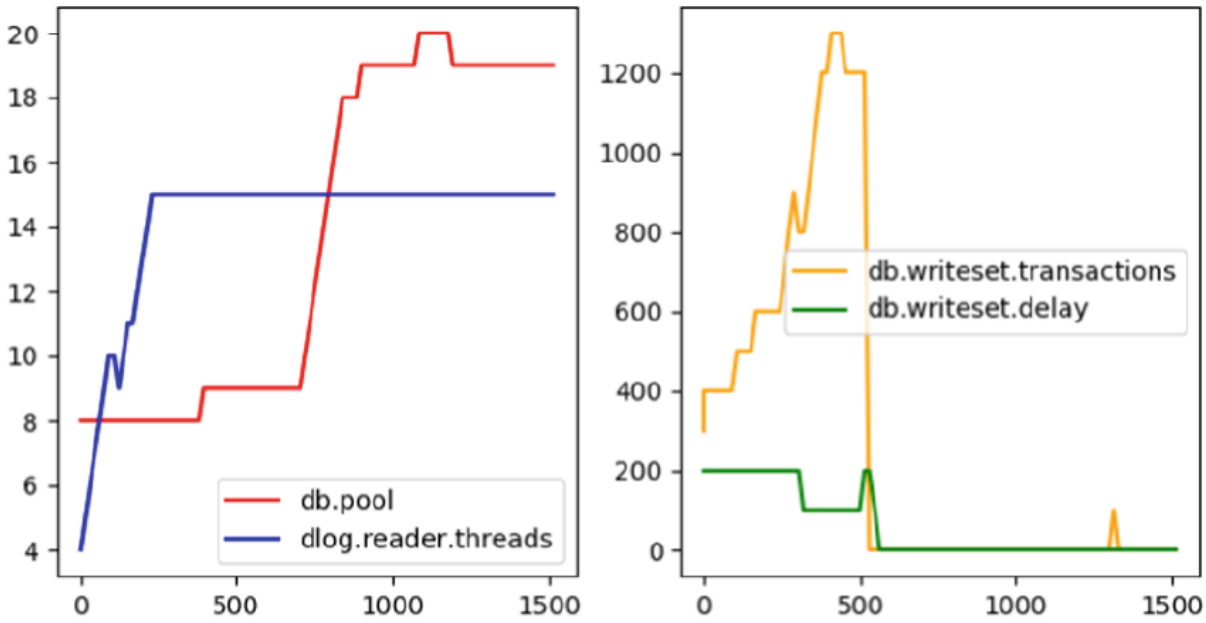
Результаты. Средние результаты являются средними результатами для каждого цикла оценки. Результаты показывают влияние на обе развернутые реплики, изображенные как R1 и R2. На всех циклах обучения производительность была лучше, чем в базовом случае, а среднее значение вознаграждения имеет тенденцию к увеличению в случае R1, в то время как в случае R2, по-видимому, достигнуто максимальный значение около 68%.

Действия, которые были предприняты при каждой корректировке RL, показаны на рис. 3.4 и 3.5. На рисунках изображены 6 из 10 циклов, подробно описывающих эволюцию каждой рассматриваемой конфигурации.

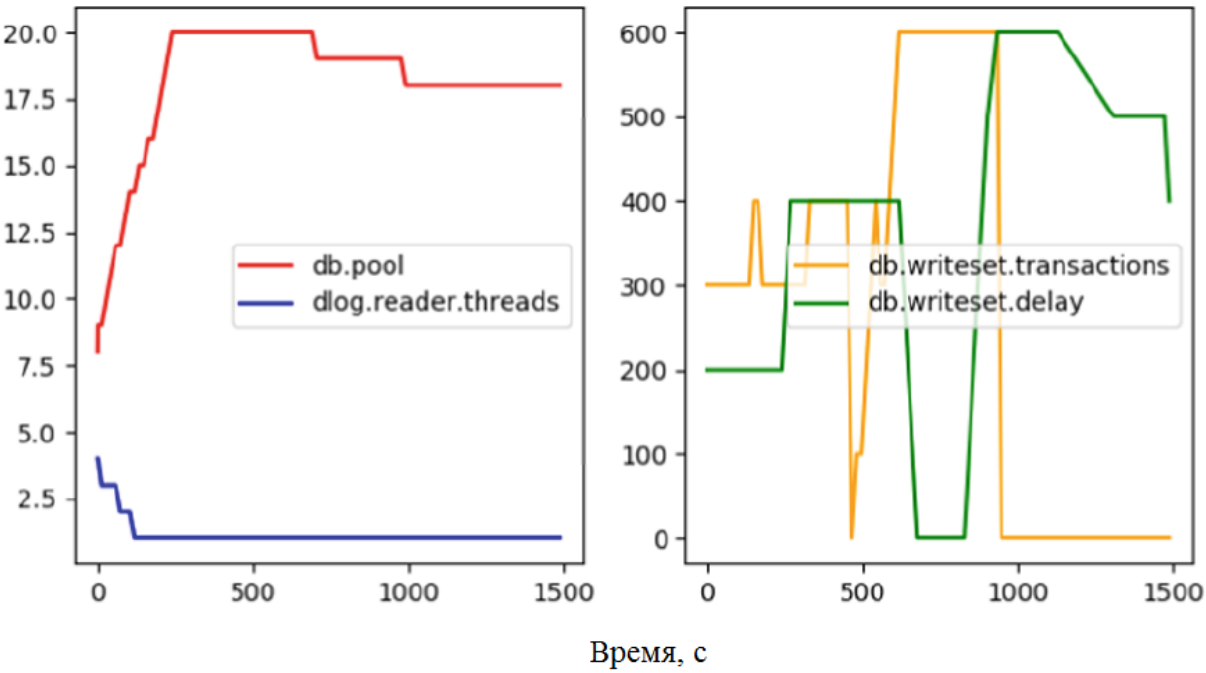
Также стоит отметить, что значение для реплицированной метрики T_{XN} одной реплики может быть только таким же высоким, как значение для метрики T_{XN}, записанной в другой реплике. Это означает, что производительность одной реплики частично зависит от производительности другой.



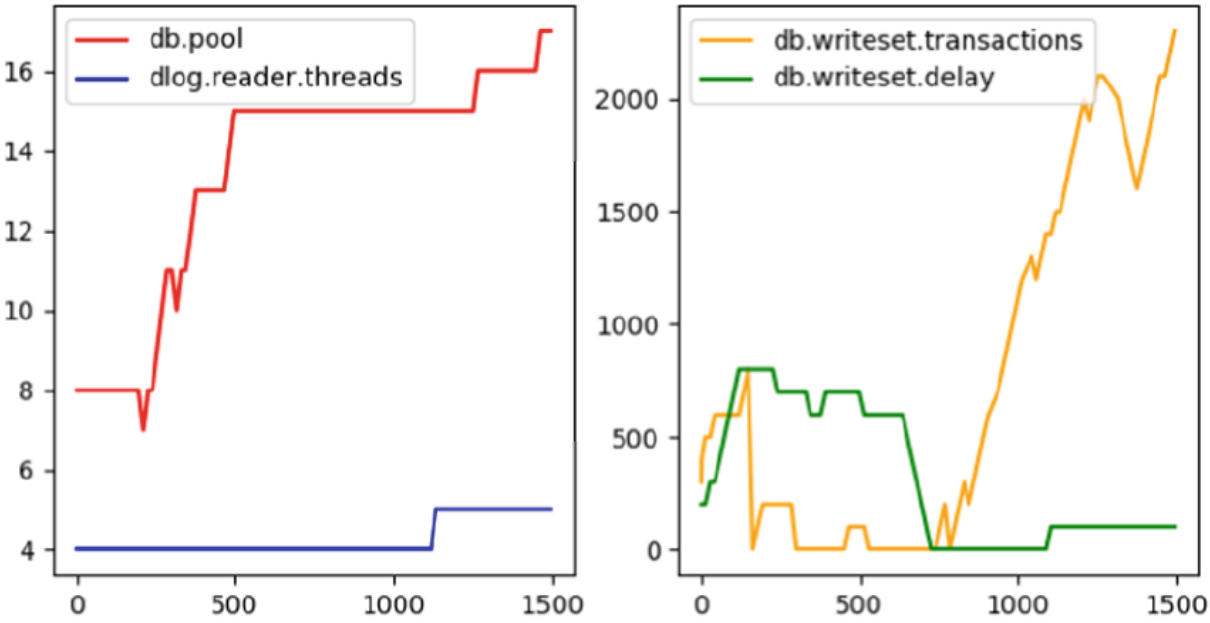
а) RL1 - Реплика 1



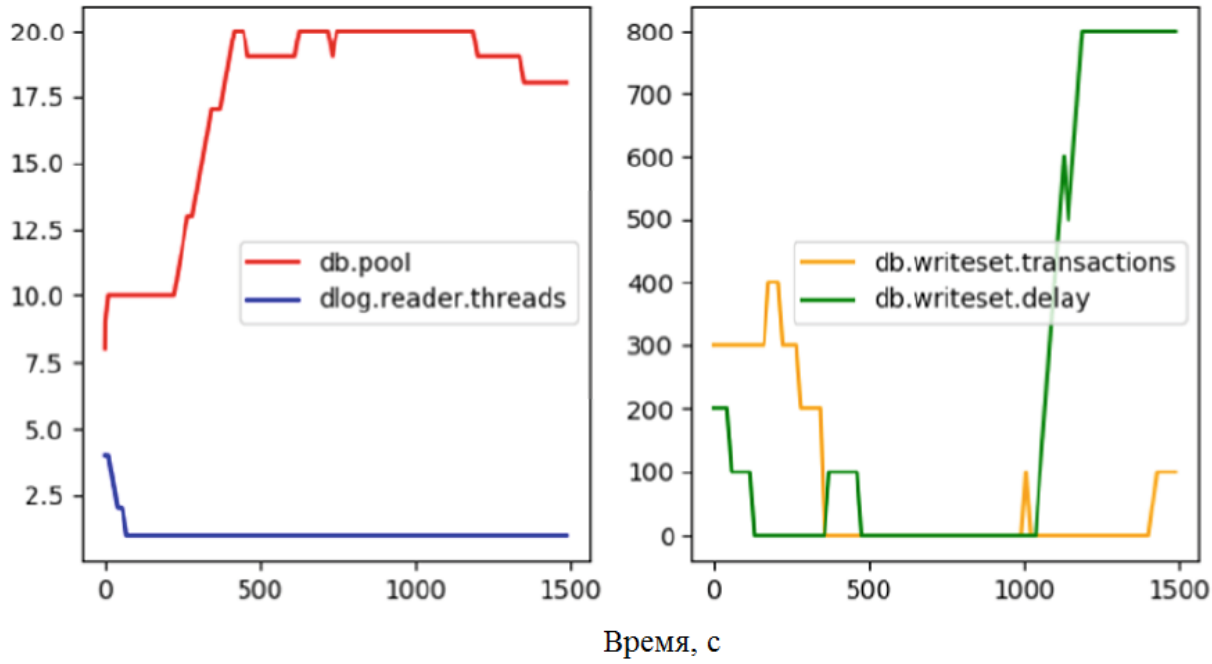
б) RL1 - Реплика 2



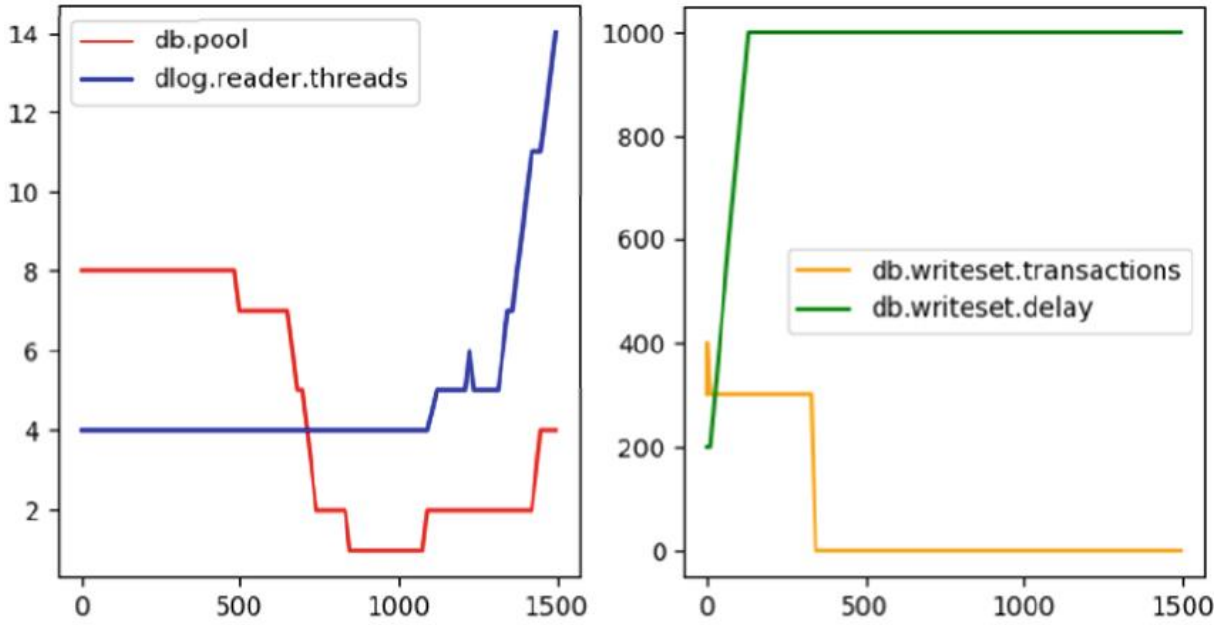
в) RL2 - Реплика 1



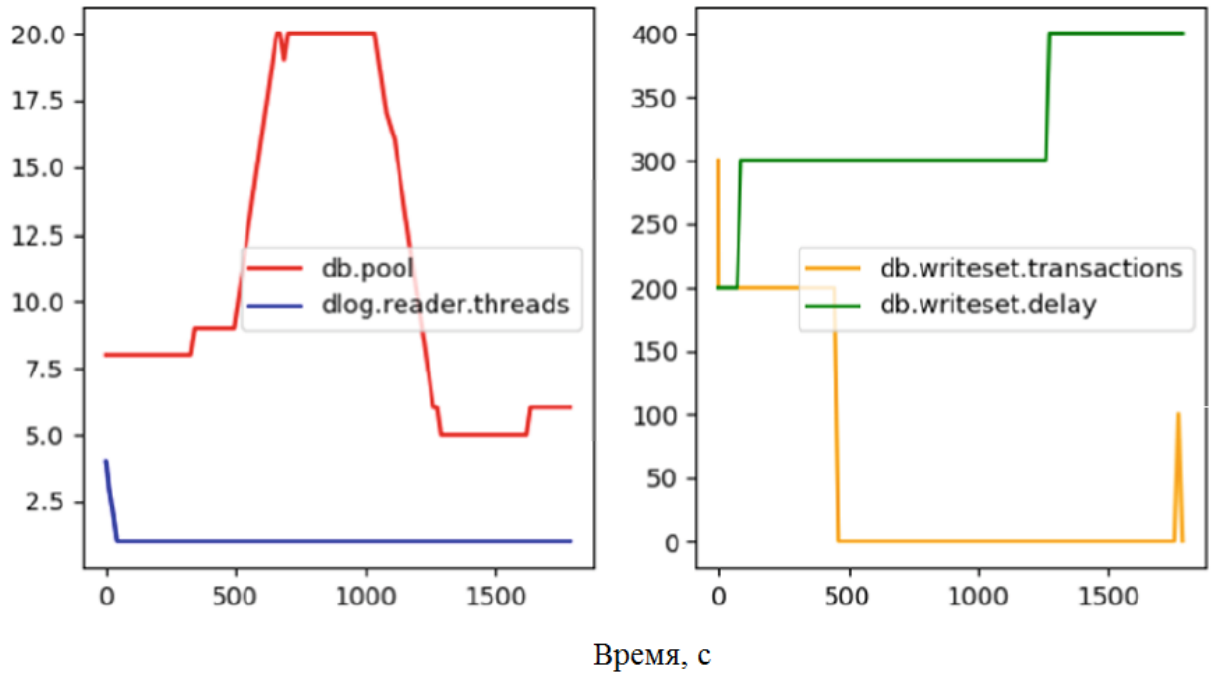
г) RL2 - Реплика 2



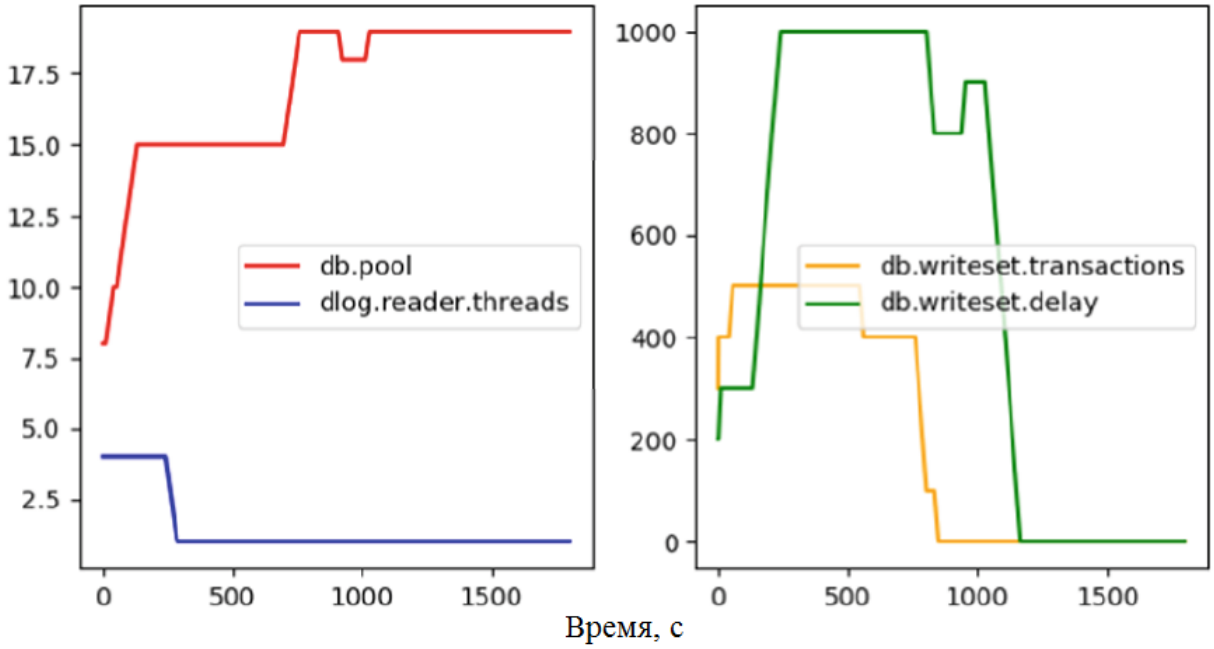
д) RL4 - Реплика 1



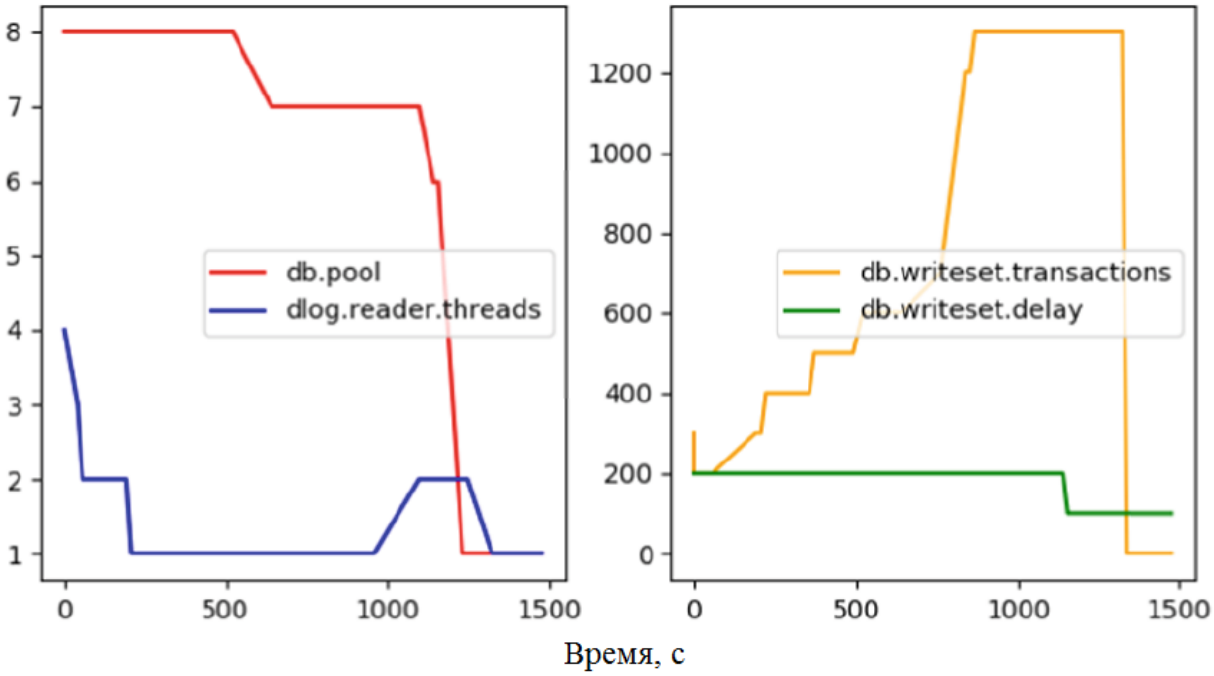
е) RL4 - Реплика 2



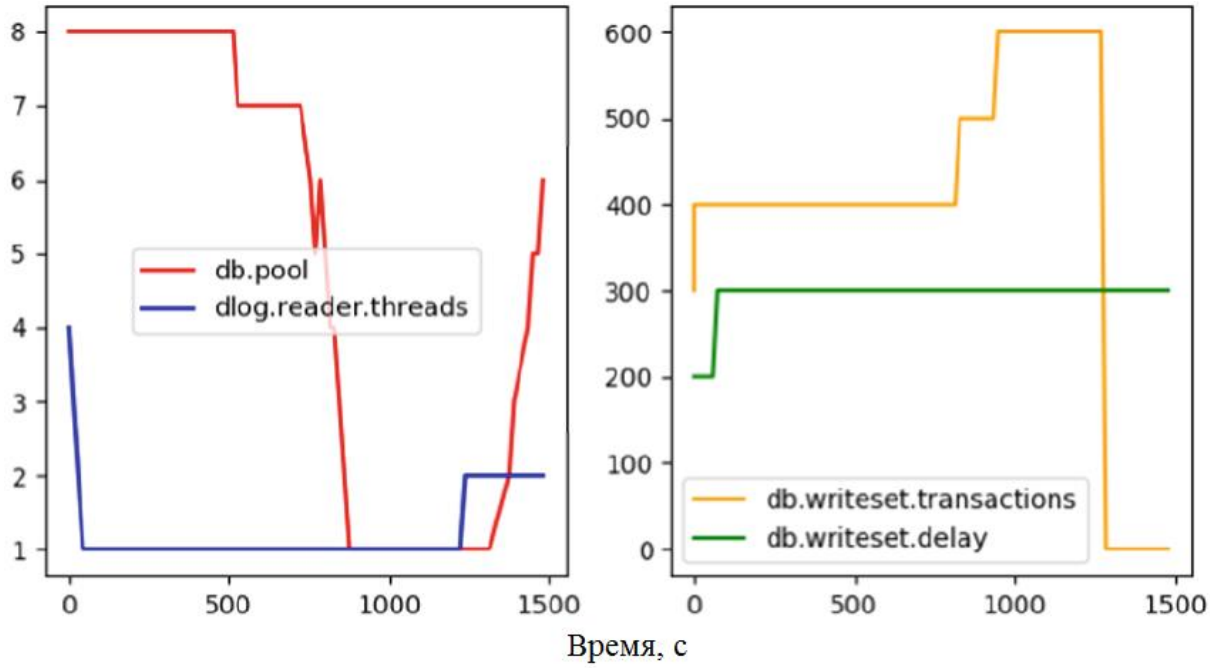
ж) RL6 - Реплика 1



з) RL6 - Реплика 2



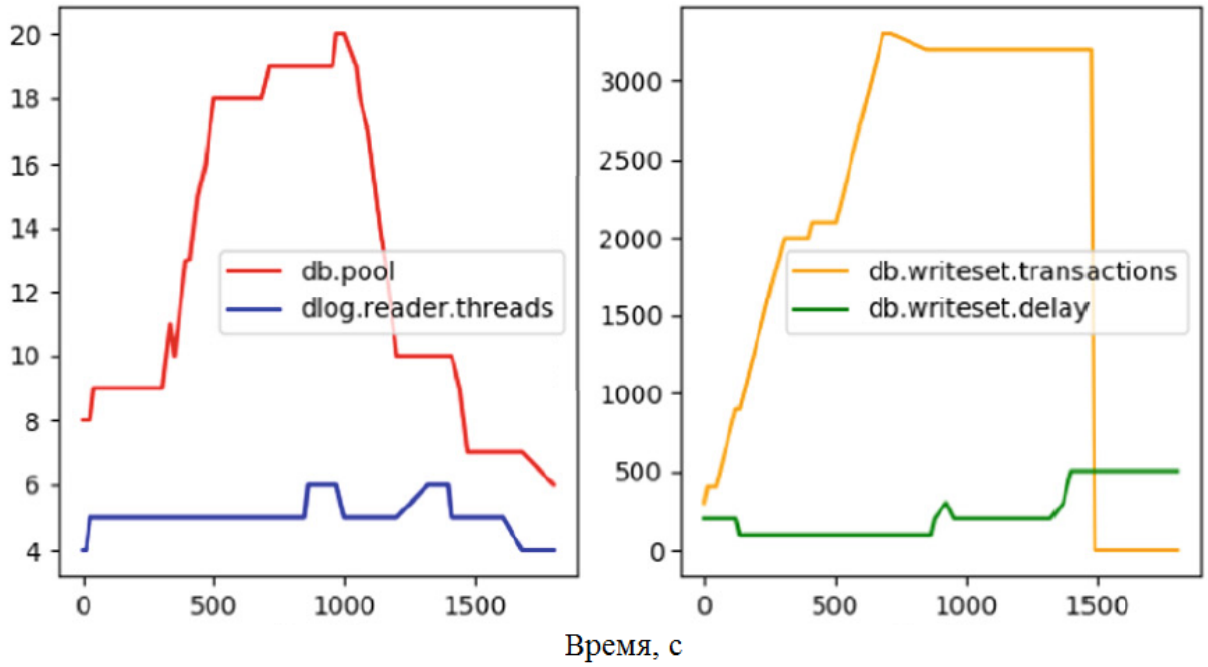
и) RL8 - Реплика 1



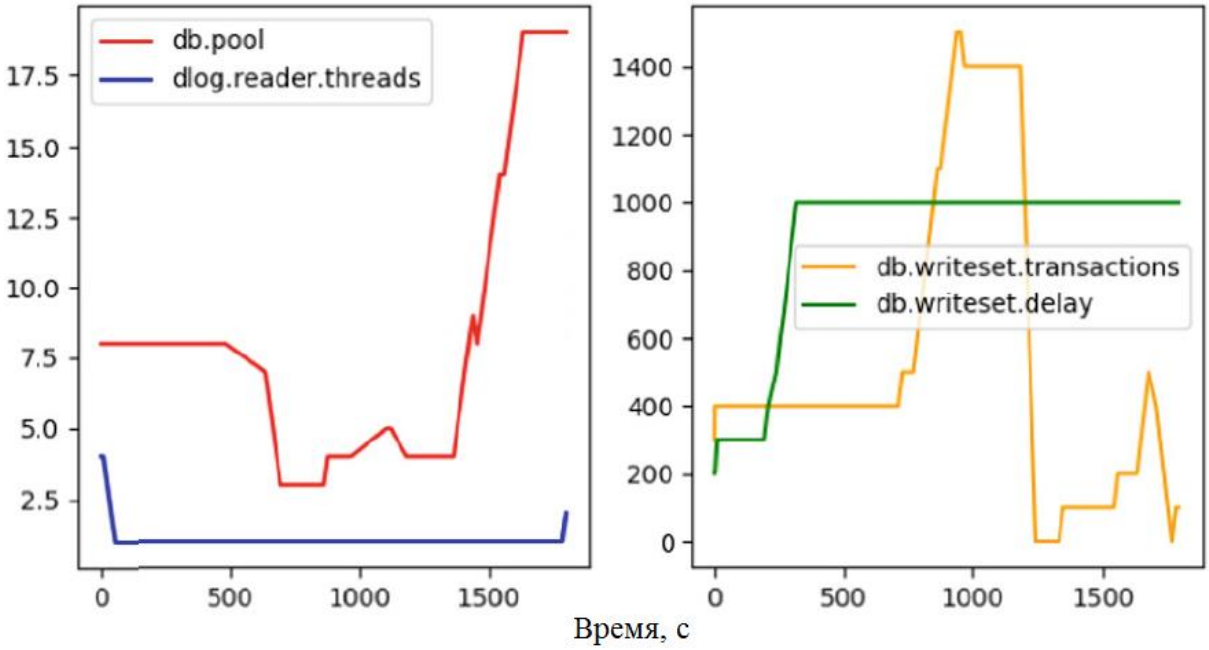
к) RL8 - Реплика 2

Рис. 3.4. Эволюция действий, предпринятых с переменными конфигурации во время сравнительного анализа, цикл 1, 2, 4, 6 и 8

Более того, более глубокий анализ изображенных результатов позволяет сделать дополнительные выводы, а именно тот факт, что во всех наблюдаемых циклах в обеих репликах в какой-то момент для db.writeset.transactions установлено значение 0. Это означает, что реплики выполнялись без ограничений для операций пакетной обработки. Более того, blog.reader.threads, который управляет пулом потоков, связанным с обработчиком чтения распределенного журнала (отвечающим за сбор данных), в основном был сброшен до минимальных значений, что подчеркивает интенсивный профиль записи рассматриваемого эталона.



а) RL10 - Реплика 1



б) RL10 - Реплика 2

Рис. 3.5. Эволюция действий, предпринятых с переменными конфигурации во время сравнительного анализа. Цикл 10

В табл. 3.5 представлены окончательные результаты, извлеченные из

табл. 3.4. В целом, сравнивая базовые результаты с последним циклом настройки, показатели состояния, непосредственно связанные с механизмом репликации, показали значительное улучшение, на порядок большее для некоторых параметров состояния.

Таблица 3.5

Обзор результатов

Metric	**Baseline (Txn/sec)**	**RL10 (Txn/sec)**	**Прирост**
ClientRequests-R1	80.80	112.95	+39.79%
Replicated Txn-R1	35.24	69.08	+96.01%
Txn Written-R1	27.57	112.95	+309.73%
ClientRequests-R2	178.20	205.47	+15.30%
Replicated Txn-R2	27.16	98.25	+261.81%
Txn Written-R2	31.86	94.26	+195.82%

Этот сценарий содержит ограниченное число переменных настройки, поэтому нейронной сети с двумя 16 скрытыми слоями было достаточно, чтобы охватить сложность проблемы.

Кроме того, состояние содержит информацию, ограниченную действием промежуточного программного обеспечения репликации, не содержащую, например, переменных, связанных с базовым оборудованием. Эти характеристики позволили обучить модель за очень короткое время, но не позволяют перенести обученную модель с одной машины на другую. Мы откладываем на будущее решение проблемы межмашинной совместимости моделей.

3.3. Проблемы и перспективы исследования

Основная работа сосредоточена на обеспечении оптимизации физического развертывания и компоновки данных систем баз данных. Связь с физической компоновкой данных позволяет системам

самонастройки ускорять создание индексов, разделов данных или материализованных представлений. Однако эти подходы не могут охватить детали, лежащие в основе СУБД, где корректировка конкретных требований к настройке влияет на внутренние компоненты СУБД. Большинство продуктов баз данных, таких как IBM BD2, Microsoft SQL Server или MySQL, включают инструменты для настройки своих соответствующих систем с учетом производительности сервера. Однако эти решения зависят от усмотрения администратора базы данных при выборе применяемых стратегий и последующих действий, предпринимаемых в СУБД.

Описанный набор инструментов также (косвенно) охватывает настраиваемые критерии, которые строго касаются настройки методов репликации, как части настраиваемых переменных конфигурации, которые доступны в каждой системе. Однако обзор настраиваемых переменных, чаще всего, отсутствует, хотя некоторые системы специализируются на выборе наиболее значимых конфигураций. Тем не менее, это, как правило, независимый процесс.

Хотя стратегии обучения с подкреплением не являются новыми в сферах машинного и автономного обучения, в настоящее время они являются популярными методами, применяемыми в различных областях вычислений. Этот класс методов охватывает агента, который пытается получить изменения состояния в соответствующей среде, стремясь при этом максимизировать баланс долгосрочной и краткосрочной отдачи функции вознаграждения. В частности, в области систем баз данных такие стратегии начали применяться во внутренних системах СУБД, чтобы помочь оптимизаторам запросов превзойти базовые стратегии оптимизации для упорядочения соединений и общего планирования запросов. Результатом обычно является последовательный набор решений, основанных на цепочках решений Маркова.

Эти методы позволяют агенту искать действия на этапе обучения, когда он осознанно предпринимает действия, чтобы узнать, как реагирует среда, или приблизиться к цели оптимизации. Более того, эти методы обычно основаны на эвристических методах исследования, основанных на статистике, собранной с окружающей среды.

Существуют решения на основе RL, специфичные для самостоятельной конфигурации базы данных с одним экземпляром. В них используют либо показатели, доступные базой данных, либо информацию из оптимизатора запросов СУБД и типа запросов, которые составляют рабочую нагрузку базы данных.

Хотя глубокое Deep Q-Learning не может работать с пространствами непрерывных действий без их выборки, в отличие от некоторых более поздних алгоритмов, таких как градиент детерминированной политики (DDPG), ему все же удается достичь очень хороших результатов. Использование Deep Q-Learning по сравнению с DDPG может быть выгодно в условиях ограниченных ресурсов, поскольку в нем используется только одна нейронная сеть вместо двух (одна для актера и одна для критика).

3.4. Выводы

Подробно описан механизм обучения с подкреплением, который подключается к промежуточному программному обеспечению репликации JDBC, оптимизируя его настраиваемые конфигурации. Представлена архитектура, демонстрирующая компоненты, которые являются частью дизайна.

Основное внимание сконцентрировано на том, что при рассмотрении механизмов самообучения, которые способны настраивать систему промежуточного программного обеспечения репликации, можно установить, что до сих пор усилия по оптимизации рассматривали

механизмы баз данных в целом и не фокусировались на конкретных внутренних компонентах, таких как контроллеры репликации. Более того, они не учитывают внешние подключаемые механизмы надежности.

Также наглядно продемонстрирована эффективность этих методов, а также основное влияние, которое регулируемые переменные настройки оказывают на общую производительность системы. Результаты подтверждают правильность подхода, подчеркивая максимальное улучшение примерно на 370,99% для некоторых из рассмотренных показателей промежуточного программного обеспечения репликации.

Список источников к главе 3

3.1. Apache Distributed Log. http://bookkeeper.apache.org/distributedlog/.

3.2. Agrawal S., Chaudhuri S., Narasayya V.R. Automated selection of materialized views and indexes in SQL databases// VLDB, 2000, pp. 496-505.

3.3. Curino C., Jones E., Zhang Y., Madden S. Schism: a workload-driven approach to database replication and partitioning// Proc. VLDB Endow., 2010, vol. 3(1-2), pp. 48-57.

3.4. Dias K., Ramacher M., Shaft U., Venkataramani V., Wood G. Automatic performance diagnosis and tuning in Oracle// CIDR, 2005, pp. 84-94.

3.5. Duan S., Thummala V., Babu S. Tuning database configuration parameters with ituned// Proc. VLDB Endow., 2009, vol. 2(1), pp. 1246-1257.

3.6. French, C.D. One size fits all" database architectures do not work for DSS// Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, pp. 449-450.

3.7. Garcıa J., Fernandez F. A comprehensive survey on safe reinforcement learning// J. Mach. Learn. Res., 2015, vol. 16(1), pp. 1437-1480.

3.8. George L. HBase: The Definitive Guide: Random Access to your Planet-Size Data. - Sebastopol: O'Reilly Media Inc., 2011.

3.9. Hintjens P. ZeroMQ: Messaging for many Applications. Sebastopol: O'Reilly Media Inc., 2013.

3.10. Li G., Zhou X., Li S., Gao B. Qtune: a query-aware database tuning system with deep reinforcement learning// Proc. VLDB Endow, 2019, vol. 12(12), pp. 2118-2130.

3.11. Marcus R., Papaemmanouil O. Deep reinforcement learning for join order enumeration// Proc. of the First Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018, pp. 3:1-3:4.

3.12. Morff A.R., Paz D.R., Hing M.M., Gonzalez L.M.G. A

reinforcement learning solution for allocating replicated fragments in a distributed database// Computacion y Sistemas, 2007, vol. 11(2), pp. 117-128.

3.13. Powell D. Delta-4: A Generic Architecture for Dependable Distributed Computing, vol. 1. Springer, Cham, 2012.

3.14. Puterman M.L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken, 2014.

3.15. Schiefer K.B., Valentin G. Db2 universal database performance tuning// IEEE Data Eng. Bull., 1999, vol. 22(2), pp. 12-19.

3.16. Schnaitter K., Abiteboul S., Milo T., Polyzotis N. Colt: continuous on-line tuning// Proc. of the 2006 ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 793-795.

3.17. Stonebraker M., Rowe L.A. The Design of Postgres, vol. 15. ACM, New York, 1986.

3.18. Storm A.J., Garcia-Arellano C., Lightstone S.S., Diao Y., Surendra M. Adaptive self-tuning memory in db2// Proc. of the 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1081-1092.

3.19. Sullivan D.G., Seltzer M.I., Pfeffer A. Using Probabilistic Reasoning to Automate Software Tuning, vol. 32. ACM, New York, 2004.

3.20. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. MIT Press, Cambridge, 2018.

3.21. Trummer I., Moseley S., Maram D., Jo S., Antonakakis J. Skinnerdb: regretbounded query evaluation via reinforcement learning// Proc. VLDB Endow., 2018, vol. 11(12), pp. 2074-2077.

3.22. Valentin G., Zuliani M., Zilio D.C., Lohman G., Skelley A. Db2 advisor: an optimizer smart enough to recommend its own indexes// Proc. of 16th Int. Conf. on Data Engineering (Cat. no. 00CB37073), 2000, pp. 101-110.

3.23. Watkins C.J., Dayan P. Q-learning// Mach. Learn., 1992, vol. 8(3-4), pp. 279-292.

3.24. Wiesmann M., Pedone F., Schiper A., Kemme B., Alonso G.

Understanding replication in databases and distributed systems// Proc. 20th IEEE Int. Conf. on Distributed Computing Systems, 2000, pp. 464-474.

3.25. Zhang J. An end-to-end automatic cloud database tuning system using deep reinforcement learning// Proc. of the 2019 Int. Conf. on Management of Data, 2019, pp. 415-432.

4. Архитектура распределенной защищенной многосерверной СУБД, разделяющая обработку транзакций и проектирование распределения вычислений и данных

4.1. Сервис-ориентированный дизайн архитектуры СУБД

Распределенная организация должна повлиять на организацию облачных СУБД. Среда облачных сервисов имеет богатый источник различных типов сбоев, таких как программное обеспечение, аппаратное обеспечение, сетевое подключение и питание, которые могут сделать службу недоступной. Сервис-ориентированный дизайн представляет собой систему слабо связанных сервисов, которые взаимодействуют друг с другом по сети, обеспечивая гибкую конфигурацию сервиса и высокую доступность. При такой конструкции разъединения вычислений и данных служба может допускать ошибки на уровне каждого компонента: замена вычислений компонент с горячим резервированием и перенаправление компонента данных на другие узлы хранения в тех же копиях. Чтобы проанализировать конструктивные соображения при использовании отделенной организации в облачной СУБД, модифицирована одна из реальных СУБД (MariaDB).

4.1.1. Внутренние компоненты MariaDB

MariaDB версии 10.1.19 и XtraDB, который является механизмом хранения данных по умолчанию для MariaDB, используется в качестве базы для прототипа. Для принятия несвязанной архитектуры внесены небольшие изменения в подсистему ведения журнала MariaDB, в то время как большинство компонентов, таких как диспетчер соединений, анализатор запросов и менеджер транзакций, остались нетронутыми. Прежде чем описать прототип, кратко опишем подсистему ведения

журнала XtraDB и ее формат журнала.

XtraDB, улучшенная версия InnoDB, которая является механизмом хранения по умолчанию для MariaDB, реализует подсистему ведения журнала с использованием глобальной блокировки. Для параллельной обработки транзакций каждая транзакция в XtraDB записывает изменения данных в локальный буфер журнала (т.е. буфер журнала мини-транзакций). Фиксация транзакции в XtraDB выполняется в два этапа: (1) добавление журналов транзакций локального буфера в буфер глобального журнала и (2) очистка глобального буфера в стабильное хранилище. При фиксации одновременно выполняемых транзакций диспетчер журналов выполняет арбитраж параллельных транзакций с использованием глобальной блокировки, тем самым сохраняя правильность ведения журнала (т.е. сериализованную очистку).

Как и другие часто используемые СУБД, XtraDB использует ОВЕН для метода ведения журнала и восстановления по умолчанию. Во время обработки транзакций XtraDB добавляет записи журнала, и каждая запись журнала содержит только байтовые (или физические) изменения, внесенные на странице. Поскольку физические записи журнала на страницу изменяются, записи журнала могут быть классифицированы на записи журнала на страницу, которые могут применяться параллельно. Кроме того, XtraDB управляет всеми данными, включая системные каталоги, в виде таблиц базы данных, и, применяя записи журнала к соответствующим страницам, можно восстановить весь образ базы данных.

4.1.2. Сервис-ориентированная архитектура защищенной СУБД

Чтобы реализовать разрозненную организацию СУБД, мы разделяем компоненты в одной MariaDB на два различных компонента обслуживания: сервер переднего плана, обеспечивающий обработку

транзакций, и серверный компонент, гарантирующий строгую защищенность и долговечность. Сервер переднего плана отправляет журналы транзакций и получает сообщения ACK от сервера внутреннего хранилища. Сообщения ACK передают надежный порядковый номер журнала (LSN), гарантирующий долговечность журналов до определенного момента. Дополнительная логика выполнения в серверной части включает в себя восстановление обновленной страницы базы данных со старой страницы, использующая эти записи журнала. Восстановленные таким образом страницы эквивалентны обновленным страницам в интерфейсе. Когда сервер переднего плана выходит из строя, служба поддержки может немедленно продолжить обслуживание с помощью предварительно восстановленных страниц или восстанавливаемых страниц по требованию. Поэтому СУБД необходимо только выполнить восстановление отмены. С другой стороны, сбои внутренних серверов могут быть обработаны другими внутренними серверами, которые совместно используют трафик журнала.

На рис. 4.1 показана общая архитектура прототипа СУБД, ориентированной на обслуживание (SO-СУБД). Два отдельных сервера соединены между собой через высокоскоростную сеть.

Поскольку обработка транзакций происходит на интерфейсном сервере, интерфейсный сервер обрабатывает запросы пользователей на обновления без каких-либо изменений в клиентском интерфейсе. Вместо этого логика выполнения, необходимая для хранения данных и журналов для обеспечения долговечности, отделяется от внешнего сервера и перемещается на внутренний сервер. Внешний сервер отправляет поток журналов транзакций на внутренний сервер, а сервер внутренний сервер отвечает сообщением ACK для LSN, который закреплен в хранилище, и представляет собой последний журнал транзакций, который становится долговечным. При реализации этой многоуровневой организации можно

рассмотреть три метода оптимизации.

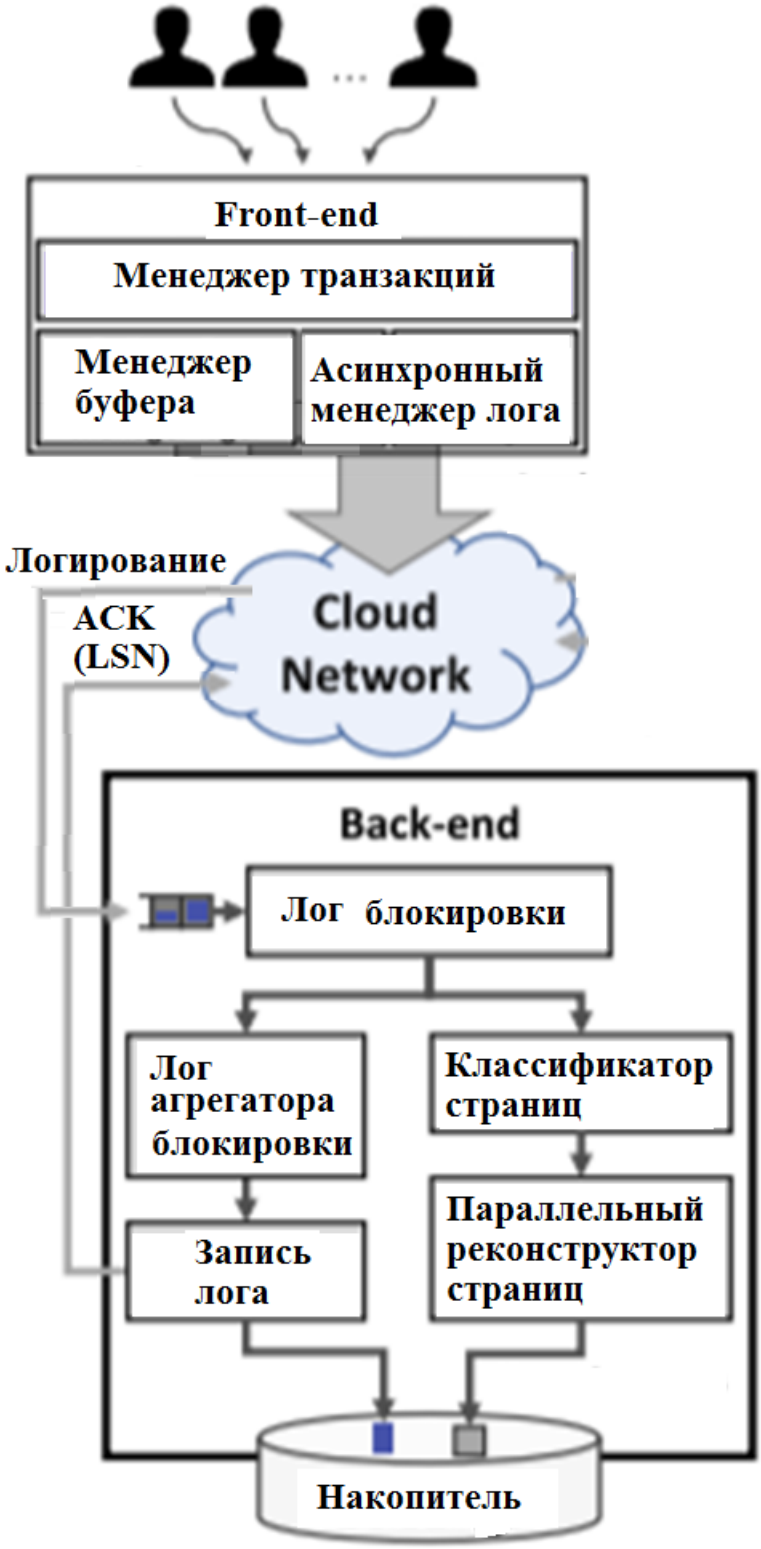


Рис. 4.1. Укрупненная структура прототипа SO-СУБД

Асинхронное ведение журнала

Подсистема централизованного ведения журнала XtraDB влияет на производительность обработки транзакций в прототипе. Задержка завершения регистрации состоит из сетевой задержки и дополнительной задержки вычислений из-за гарантии долговечности журналов транзакций в серверной части. Чем дольше задержка пути ввода-вывода во время фиксации транзакции, тем ниже пропускная способность транзакции. Поскольку базовая линия использует грубую синхронизацию поверх ведения журнала, одновременное выполнение транзакций ограничено из-за длительной задержки этого синхронного пути.

Восстановление страницы

База данных обычно использует ведение журнала с опережением записи (или WAL) для гарантии атомарности транзакций. WAL требует дважды записывать данные для каждого изменения записи: один для журнала, а другой для соответствующей страницы. На рис. 1.3 показано общее количество записей для ведения журнала и очистки страниц в XtraDB. Размер страницы является основным фактором усиления записи. Таким образом, когда СУБД использует больший размер страницы, требуется записать больше данных. Характеристики рабочей нагрузки также влияют на общий объем операций записи. Равномерная рабочая нагрузка, серая полоса на графике, записывает больше данных, чем искаженная рабочая нагрузка. Когда СУБД использует сетевое хранилище, это увеличивает сетевой трафик. Серверная часть может уменьшить сетевой трафик при сбросе страниц, заменив журналы повтора, вместо переноса всех обновленных страниц.

Эффективное восстановление страницы

Дизайн OBHA позволяет выполнять параллельное восстановление

страниц с использованием журналов транзакций на каждой странице, что может ускорить обработку повторного воспроизведения. XtraDB использует ведение журнала транзакций в стиле ARIES, и его байтовое ведение журнала в каждой записи журнала не раскрывает межстраничные зависимости. Каждая история обновлений для каждой страницы обеспечивает правильное восстановление страницы, поскольку она сохраняет частичный порядок внутри страницы. Между тем, общая пропускная способность системы часто ограничивается самым медленным компонентом в системе, и восстановление страницы может привести к системным задержкам из-за ввода-вывода страницы. Для устойчивой системы производительность, следовательно, необходимо учитывать эффективную конструкцию перестройки страницы.

Прототип SO-СУБД

Реализован прототип SO-СУБД поверх MariaDB. В нашем прототипе регистрация в подпрограмме фиксации транзакций XtraDB может быть оптимизирована с помощью асинхронного метода ведения журнала. Затем мы обсудим важные детали реализации нашего прототипа: (1) совершение асинхронной операции, (2) параллельное восстановление страницы и (3) рабочая модель восстановления страницы.

4.1.4. Совершение асинхронной операции

Чтобы обеспечить долговечность, фиксируемая транзакция не может завершиться до тех пор, пока журналы транзакций не будут сохранены в стабильном хранилище. При использовании сетевого хранилища эта процедура синхронной фиксации может серьезно повлиять на производительность системы из-за увеличения сетевой задержки. Поэтому прототип SO-СУБД обрабатывает фиксацию транзакции асинхронным способом. На рис. 4.2 представлена асинхронная фиксация транзакции.

Рис. 4.2. SO-DBMS компоненты

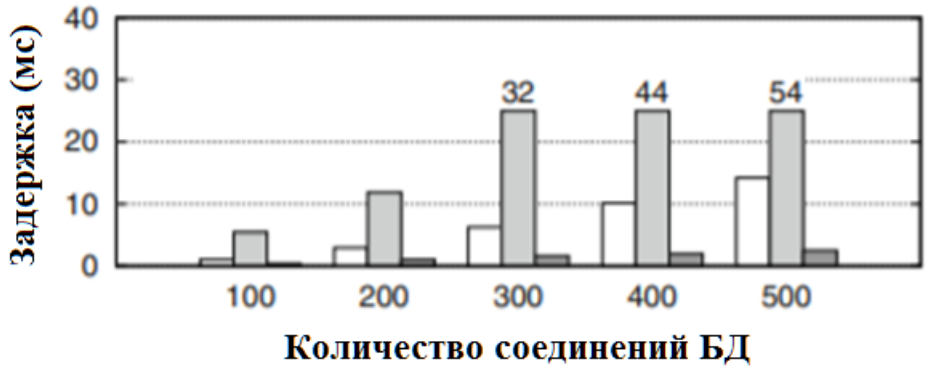
Каждая транзакция сначала резервирует LSN для своих журналов, а затем копирует локальные журналы транзакций в глобальный буфер журналов. Журналы в глобальном буфере отправляются во внутренний сервер. Поскольку обработка транзакций в XtraDB реализует двухфазную блокировку (2PL), она освобождает блокировки и другие ресурсы,

связанные с текущей транзакцией, только после завершения регистрации. Длительная задержка завершения регистрации в транзакции способствует серьезному конфликту блокировок.

Скрытие задержки

На рис. 4.3 сравнивается задержка завершения ведения журнала при использовании различных типов хранилища. СУБД, использующие сетевое хранилище, увеличивают время обработки транзакций до 59 раз по сравнению с локальным хранилищем.

а) задержка завершения ведения журнала



б) возрастание продолжительности фазы в протоколе двухфазной фиксации

Рис. 4.3. Задержка завершения ведения журнала влияет на время обработки транзакций при запуске sysbench benchmark (OLTP): ▭ - Maria DB SSD; ▭ - Maria DB NFS; ▭ - Прототип SO-СУБД

Завершение регистрации в SO-СУБД добавляет дополнительную

задержку процедуры проверки блоков, а SO-СУБД с существующим протоколом фиксации транзакций еще больше увеличит задержку обработки транзакций. Вместо того, чтобы ждать сообщения ACK для надежных журналов, каждая транзакция регистрирует свой собственный контекст в ожидающем очередь завершения и оставляет ресурсы для выполнения (например, глобальную блокировку) для других транзакций, готовых к выполнению [4.1]. Выделенный прослушиватель ACK подтверждает LSN в ответе и завершает незавершенные транзакции до этого LSN в очереди ожидающего завершения. Однако две основные спорные части в централизованной структуре подсистемы ведения журнала XtraDB могут мешать друг другу, и пропускная способность транзакций может быть снижена при больших рабочих нагрузках транзакций. Децентрализованный дизайн подсистемы ведения журнала оставлен для будущей работы.

Долговечность уровня блоков для журналов

XtraDB использует метаданные уровня блоков, которые изначально предназначены для корректности журналов при использовании блочных устройств, и поэтому СУБД использует их при регистрации в сети. Журналы в виде сетевых пакетов объединяются в серию блоков журналов. Средство проверки блоков журналов на внутреннем сервере проверяет каждый заголовок блока на предмет завершения всех журналов транзакций. Тем временем записывающий журнал на внутреннем сервере сбрасывает каждый блок журнала в стабильное хранилище и отвечает интерфейсному серверу сообщением ACK с номером LSN последнего журнала транзакций в блоке. Пока отправляя сообщение ACK для надежного блока журнала, записывающий журнал отмечает точку резервного копирования на основе порога. До этого момента все восстановленные страницы сбрасываются в хранилище.

4.2. Параллельное восстановление страницы

Рис. 4.2 иллюстрирует параллельное восстановление страницы. Каждая запись в журнале в XtraDB содержит код операции и соответствующий идентификатор страницы. Каждый код операции описывает тип операции и расположение физической записи в записи журнала. Классификатор страниц проверяет код операции каждой записи журнала, чтобы определить идентификатор страницы и положение следующей записи журнала. Затем запись в журнале добавляется в журнал обновлений для каждой страницы. Индекс журнала страниц, который сопоставляет идентификатор страницы с соответствующей историей обновлений для каждой страницы, совместно используется классификатором страниц и восстановителем страниц. Страницы могут быть реконструируется параллельно, заменяя эти независимые истории обновлений на странице. Восстановление страницы часто включает в себя ввод-вывод страниц: загружает несуществующие страницы данных в буфер страниц и удаляет «грязные» страницы, версия которых превышает точку резервного копирования, на которой отмечена запись журнала. Для минимального вмешательства ввода-вывода страницы в восстановление страницы средство восстановления страницы асинхронно сбрасывает завершенную страницу.

4.3. Восстановление страницы рабочей модели

Реконструктор страниц может столкнуться с проблемой масштабируемости по мере роста числа производителей страниц. В то время как каждый сервер выполняет процедуру выполнения независимо, эти сервери могут совместно использовать сложные подсистемы СУБД (например, менеджер буферов) или другие глобальные ресурсы (например, очередь заданий и сопоставление страниц с файлами). Кроме того, поскольку только одному серверу одновременно разрешено применять

журналы на странице, для поддержания согласованного состояния страницы между серверами требуется синхронизация. Каждая процедура восстановления страницы не зависит от данных. Поэтому для этого не требуется никакой абстракции транзакций или другой координации подсистемы СУБД. Назначение страниц базы данных выделенным сотрудникам может устранить ненужные накладные расходы на связь или переназначение между серверами.

На рис. 4.4 сравниваются две конструкции реконструктора страниц, использующие назначение "поток-работа" и "поток-данные" соответственно.

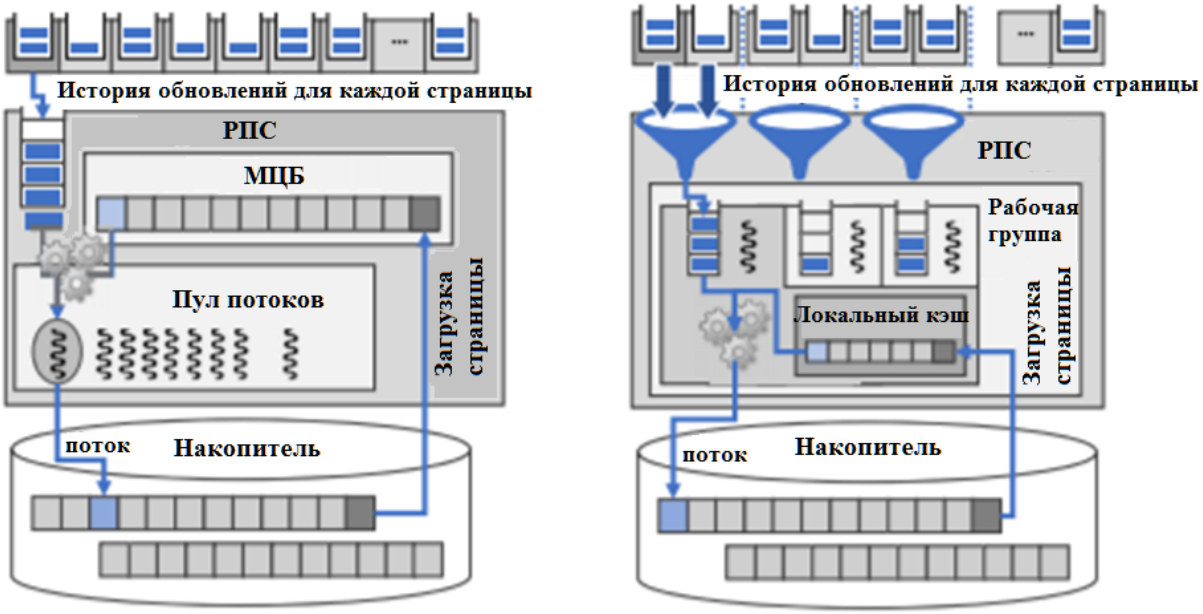


Рис. 4.4. Варианты проектирования для параллельного восстановления страницы: (1) назначение потока на работу: совместное использование очереди заданий и менеджера буферов, (2) назначение потока на данные: целевое назначение очереди заданий и менеджера буферов для каждого сервера; РПС – реконструктор параллельных страниц; МЦБ – менеджер централизованного буфера

Сквозной дизайн реализует глобальную очередь заданий, которая используется для назначения заданий на восстановление страниц серверам.

Каждый сервер использует общий менеджер буферов, который обрабатывает страницы iOS. С другой стороны, при проектировании "Поток-данные "выделяются вычислительные ресурсы, такие как очередь заданий, менеджер буферов и очистка страниц для каждого сервера. Поскольку каждый сервер делится индексом страницы только с классификатором страниц способом "один писатель - несколько читателей", координация между серверами сведена к минимуму. Страницы базы данных разделяются, и задание по восстановлению новой страницы назначается выделенному исполнителю.

4.4. Экспериментальная оценка

Рассмотрено влияние сервис-ориентированного дизайна на облачную архитектуру базы данных. Хотя сервис-ориентированный дизайн облачной базы данных нацелен на высокую доступность, службы СУБД также предъявляют высокие требования к производительности. В сервис-ориентированной организации компоненты СУБД взаимодействуют друг с другом по сети, и такое поведение может повлиять на производительность системы. Между тем, поскольку SO-СУБД реализует конвейерную архитектуру, самый медленный компонент может ограничить общую пропускную способность системы или потребовать значительно большого буфера способность сокращать разрыв в производительности между компонентами. Изучим эффективность канала для обеспечения стабильной работы системы. В этом разделе описывается поведение обработки транзакций нашего прототипа SO-СУБД с различными рабочими нагрузками.

4.4.1. Условия эксперимента

Все эксперименты в этом разделе выполняются на двух серверных машинах, соединенных друг с другом сетью с каналом 10 ГБ. Внешний

интерфейс оснащен четырьмя 18-ядерными процессорами Intel Xeon E7-8870 v3 с 256 ГБ памяти, а внутренний - четырьмя 8-ядерными E7-8837 с 128 ГБ памяти. Оба сервера оснащены локальным и сетевым SSD-накопителем Samsung 850 Pro для устройств хранения данных. Также используется один дополнительный сервер для эталонного клиента. Все серверы отключают гиперпоточность для оценки.

4.4.2. Производительность

В этом разделе анализируется производительность обработки транзакций с использованием sysbench [4.2] и TPC-C [4.3]. Сравнивается прототип SO-СУБД с MariaDB. Базы данных настроены на страницу данных объемом 4 КБ, пул буферов объемом 32 ГБ и минимальный уровень изоляции. На рис. 4.5 показана пропускная способность по оценке sysbench баз данных с различным числом клиентских подключений. В то время как количество клиентских подключений увеличивается с 50 до 500, сравнивается прототип SO-СУБД с Vanilla MariaDB, который использует либо локальные устройства хранения данных, либо сетевое хранилище более 1 или 10 Гб сетевого соединения. Система асинхронного ведения журнала SO-СУБД повышает пропускную способность лучше, чем две другие версии систем Vanilla. Однако увеличение числа клиентских подключений не приводит к увеличению производительности обработки транзакций. Самый медленный компонент в конвейерной архитектуре серверной части часто ограничивает общую пропускную способность. Анализируется компонентная пропускная способность SO-СУБД в разделе 4.4.4.

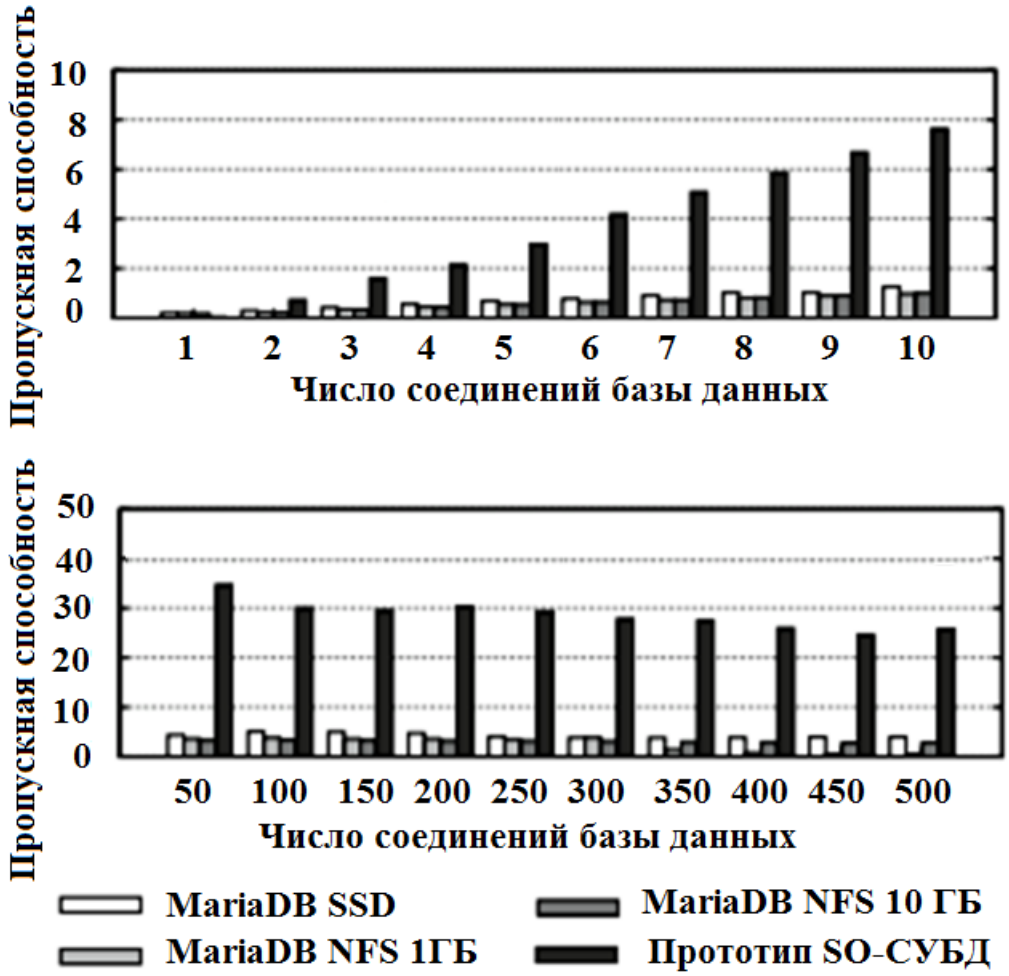


Рис. 4.5. Сравнение пропускной способности обработки транзакций по сравнению с эталоном sysbench. Работа ведется с локальными записями размером 128 байт в собственной таблице. График вверху – пропускная способность в интервале от 1 до 10 соединений; график внизу - пропускная способность в интервале от 50 до 100 соединений

На рис. 4.6 показана производительность обработки транзакций TPC-C для разного количества хранилищ. Каждый клиент настроен на доступ к одному хранилищу. В этой рабочей нагрузке SO-СУБД работает в 1,99 раза лучше, чем MariaDB с локальным хранилищем SSD. Чем больше клиентских подключений будет введено, тем хуже будет снижение производительности. Производительность снижается в основном из-за узких мест доступа на горячих системных страницах (например, на

страницах заголовка для пространства журнала). В отличие от показателя sysbench, клиенты в бенчмарке TPC-C обычно обновляют одно и то же таблицы и синхронизация в этих таблицах могут ограничивать степень параллелизма.

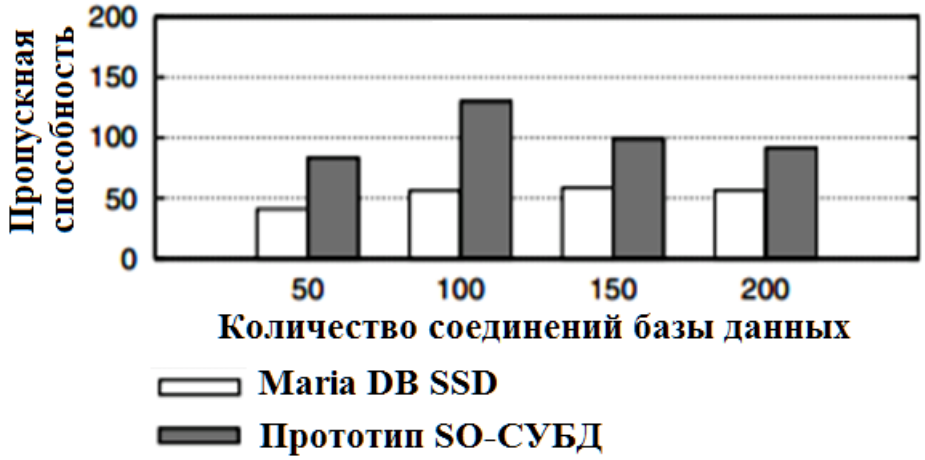


Рис. 4.6. Сравнение пропускной способности обработки транзакций по сравнению с эталоном TPC-C. Каждый сервер транзакции назначается своему собственному хранилищу, но использует общие таблицы

4.4.3. Задержка фиксации транзакции

Проанализируем влияние подхода асинхронного ведения журнала на среднюю задержку фиксации. Во время асинхронного ведения журнала подсистема ведения журнала не ожидает завершения очистки журнала, но позволяет рабочему потоку базы данных выполнять другие транзакции для лучшей загрузки ЦП.

На рис. 4.7 (верх) показано, как средняя задержка фиксации на транзакцию компенсируется подсистемой асинхронного ведения журнала. Для СУБД SO очистка журналов по сети приводит к сетевой задержке и накладным расходам на обработку для проверки записей журнала. При наличии одного клиентского подключения подсистема ведения журнала не может скрыть эту дополнительную задержку во время сброса журнала по сети, но по крайней мере два клиентских подключения могут скрыть

большую часть этой задержки. Средняя задержка фиксации транзакций уменьшается до постоянного времени, и это позволяет линейно масштабировать пропускную способность транзакций.

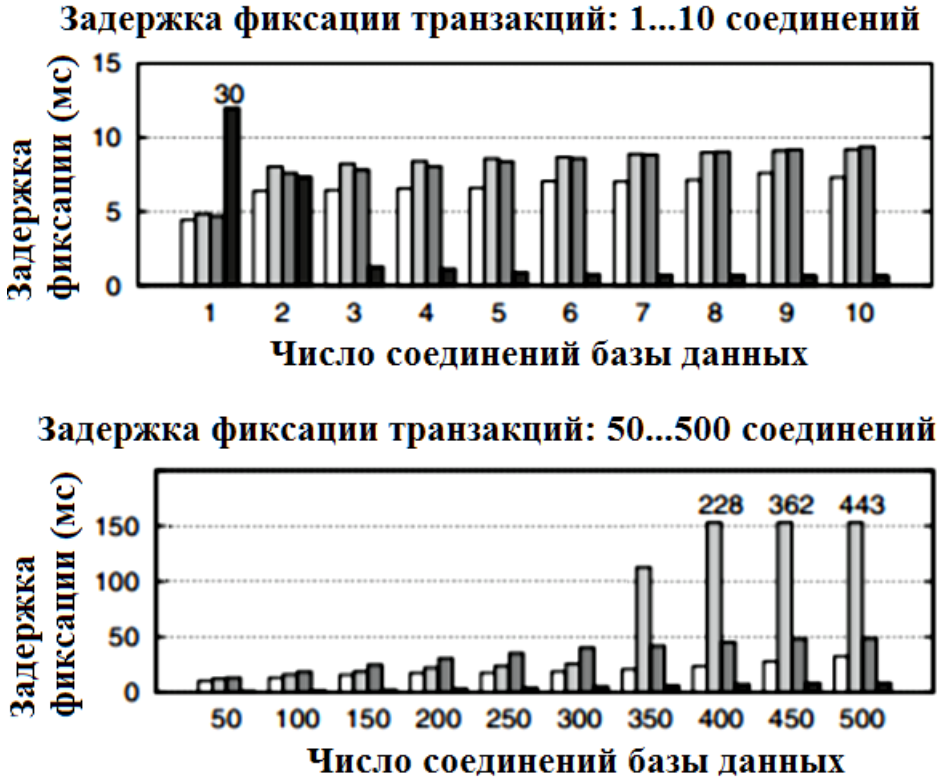


Рис. 4.7. Асинхронное ведение журнала уменьшает среднюю задержку фиксации на транзакцию

На рис. 4.7 (низ) средняя задержка фиксации SO-СУБД постепенно увеличивается по мере увеличения числа клиентских подключений, но по-прежнему показывает меньшую задержку по сравнению с двумя другими версиями. Две разные конфигурации NFS показывают более длительную задержку, что добавляет сетевую задержку к процедуре синхронного завершения ввода-вывода. Особенно с учетом конфигурация сети 1 ГБ, MariaDB показывает значительное увеличение задержки фиксации транзакций, когда количество подключений превышает 300. Для большого числа подключений эта конфигурация показывает более длительную транзакцию на 109 произошедших задержек. Конфликт ввода-вывода

между ведением журнала и сбросом «грязных» страниц в сети с ограниченной пропускной способностью 1 Гб увеличивает задержку фиксации, а длительное время выполнения в фазе роста 2PL увеличивает конфликт блокировок.

4.4.4. Использование ЦП

Поскольку СУБД имеет разобщенную организацию, которая разделяет компоненты СУБД на разные серверы, требуется дополнительный процесс для проверки блоков журналов, классификатора страниц и восстановления страниц на сервере обратной отправки. Изучим вычислительные ресурсы, необходимые для этого процесса.

На рис. 4.8 показано влияние различного количества серверов по восстановлению страниц на пропускную способность транзакций. Каждый эксперимент проводился с использованием показателя sysbench с 200 клиентских подключений. Как показано на рис. 4.8 (верх), до четырех серверов по восстановдению страниц могут повысить пропускную способность в текущей системе. Чтобы оценить влияние на производительность большего числа серверов по восстановлению страниц, мы анализируем загрузку ЦП для каждого эксперимента. На рис. 4.8 (низ) показана разбивка загрузки ЦП и доля каждого компонента на внутреннем сервере. Увеличение числа серверов по восстановлению страниц до 4 может увеличить долю полезных операций, т.е. восстановление страниц. Однако использование более четырех серверов только увеличивает часть ядра и не улучшает пропускную способность. В конвейерной архитектуре внутреннего сервера, по крайней мере, четыре сервера могут достичь максимальной пропускной способности одного классификатора страниц.

Рисунок 4.8. Влияние на производительность обработки транзакций при различном количестве построителей страниц

4.4.5. Эффективность конвейерной архитектуры

Итак, прототип СУБД требует, чтобы каждый компонент в конвейере работал эффективно. В противном случае общая производительность системы ограничивается самым медленным компонентом. Чтобы оценить эффективность конвейера, мы измеряем пропускную способность каждого компонента в серверной части и

сравниваем влияние на производительность различных схем назначения потоков. В качестве эталона для этой цели запускается микро-бенчмарк, который воспроизводит 10 ГБ потока журналов, записанных из показателя sysbench, с 500 клиентскими подключениями.

На рис. 4.9 показано влияние различных схем назначения потоков на производительность восстановления страниц.

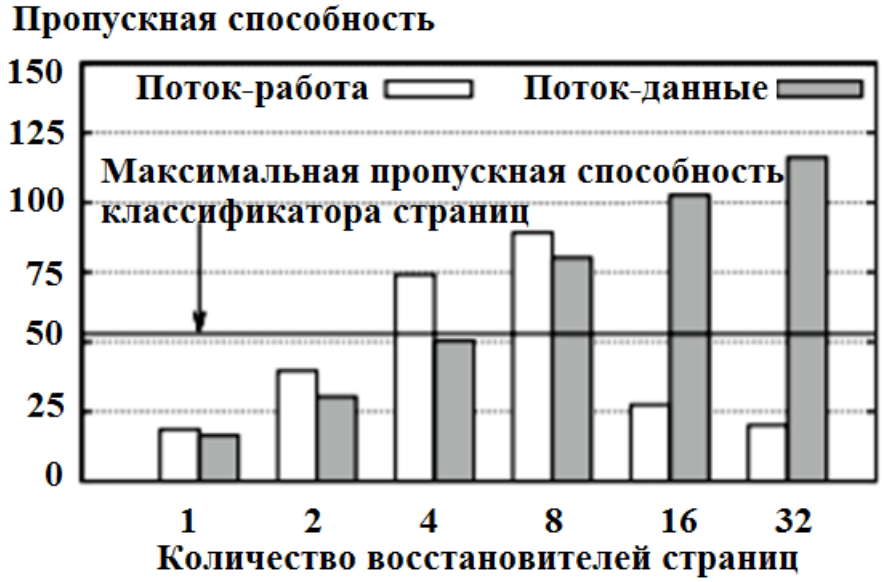


Рис. 4.9. Выполнение восстановления страниц в двух различных схемах назначения потоков с различным количеством серверов. Подход "поток-данные" обеспечивает масштабируемую пропускную способность по сравнению с подходом "поток-работа". Предыдущий конвейер классификатора страниц ограничивает скорость ввода в 66,9 Мбит/с (отмечено на графике), где общая пропускная способность системы в фоновом режиме будет ограничена

Как для подхода "поток-работа", так и для подхода "поток-данные" пропускная способность восстановления страниц увеличивается по мере увеличения числа серверов до 8. При подходе "поток-работа" каждому серверу требуется проконсультироваться с менеджером буферов для обеспечения согласованного состояния страницы. Централизованный менеджер буферов, даже несмотря на то, что он использует несколько

экземпляров для балансировки нагрузки и уменьшения конкуренции, сопряжен с ненулевыми затратами на синхронизацию. По мере увеличения числа серверов любой централизованный ресурс может стать узким местом масштабируемости. Вместо этого, используя подход "поток-данные", выделенный менеджер буферов устраняет любые накладные расходы на синхронизацию или координацию между серверами. Подход "поток-данные" в SO-СУБД использует только тонкий слой сопоставления страниц с файлами и поддерживает масштабируемость с большим количеством серверов, чем подход "поток-работа". Однако простой кэш LRU не поддерживает асинхронное чтение страницы или предварительную выборку и часто блокирует выполнение работы во время загрузки страницы. Несбалансированное рабочее задание также увеличивает критический путь при параллельном восстанодении страницы.

На рис. 4.10 показан углубленный анализ снижения производительности при параллельном восстановдении страницы путем учета использования ЦП для функций блокировки. Подход "поток-работа" тратит больше процессорного времени на функции блокировки при обработке с большим количеством серверов. С другой стороны, он обходится без каких-либо функций блокировки или синхронизации в процедуре восстановления страницы; он обеспечивает минимизацию процессорного времени функций блокировки.

Распределение обязанностей

Подход "Поток-данные" в серверной части следует за разъединением физических данных, и это может привести к несбалансированному назначению заданий. На рис. 4.11 показано распределение заданий по количеству заданий или общему размеру обновлений страниц для 32 сервера. Поскольку тест sysbench имеет единый шаблон доступа в качестве характеристики рабочей нагрузки, текущая реализация, использующая

назначение заданий на основе хэша, не раскрывает сильно несбалансированное назначение заданий. Слева показано количество обновлений, отсортированное по числу помещений в очередь, справа – по размеру записей лога.

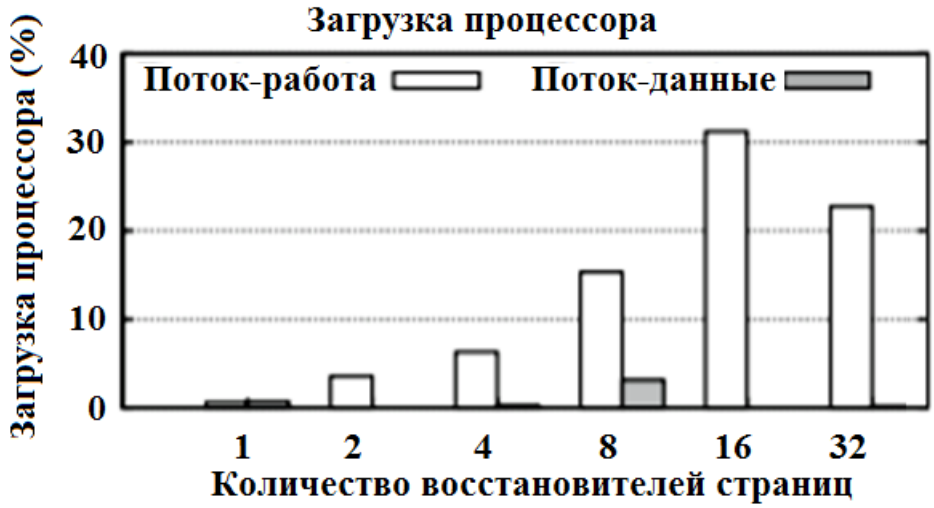


Рис. 4.10. Накладные расходы на синхронизацию, полученные из функций блокировки при предыдущей рабочей нагрузке микро-бенчмарка: пропускная способность восстановления страниц при подходе "поток-работа" снижается более чем у 8 серверов из-за относительно более высоких накладных расходов на синхронизацию, чем у других.

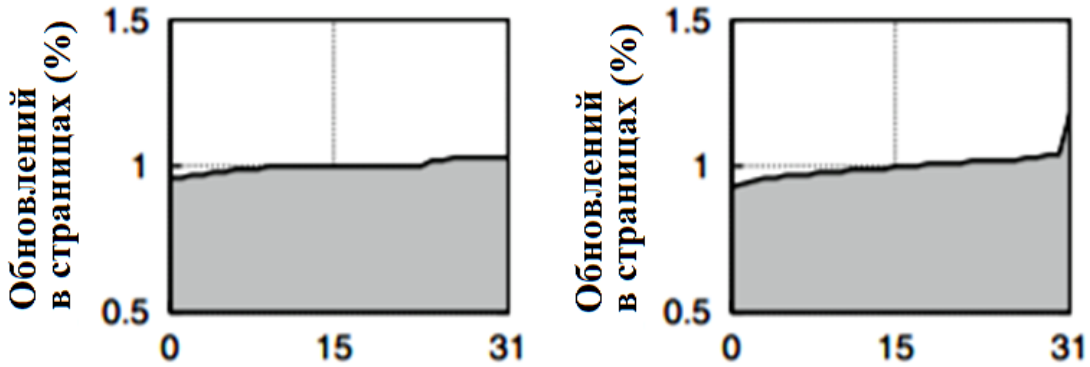


Рис. 4.11. Распределение заданий по количеству назначений и размеру записей журнала в рамках рабочих нагрузок sysbench. Результат показывает, что эта рабочая нагрузка не приводит к серьезному несбалансированному распределению заданий

4.5. Человеко-машинная система проектирования базовой структуры БД

На проектном этапе работы с СУБД важным компонентом является проектирование ER-диаграмм. В работе создана человеко-машинная система, автоматизирующая этот процесс и интегрирующая сведения о СУБД. Разработанная система позволяет получать ER-диаграммы на основе созданных или загруженных из баз данных таблиц, просматривать sql код, строить связи между ними и осуществлять вывод полученных результатов на печать, в виде отчета, применять разнообразное графическое оформление к объектам диаграмм

4.5.1.Структура системы

Система состоит из 27 основных форм (Borland Delphi). Все классы наследованы от класса TForm – визуального класса, соответствующего объекту Window ОС Windows, содержащие различные визуальные компоненты. Для доступа к данным используется механизм ADO.

На рис. 4.12 отражена упрощенная модульная структура программы, в которой отражены только те модули, в которые вносились изменения при разработке. Структура модулей программы приведена на рис. 4.13. В модули, отмеченные на рисунке сплошной заливкой, автором были внесены изменения для выполнения поставленной задачи. Модули обозначенный пунктирной заливкой, разработаны автором лично.

Модульная структура программы включает в себя 24 основных модуля. Каждый модуль несет ответственность за выполнение определенных операций, обеспечивая целостность работы. В каждом модуле реализованы функции, соответствующие его названию. Часть функций программы, не требующих отдельных форм, реализованы внутри модуля MEMMain, в частности функция удаления ролей, пользователей,

учетных записей и др.

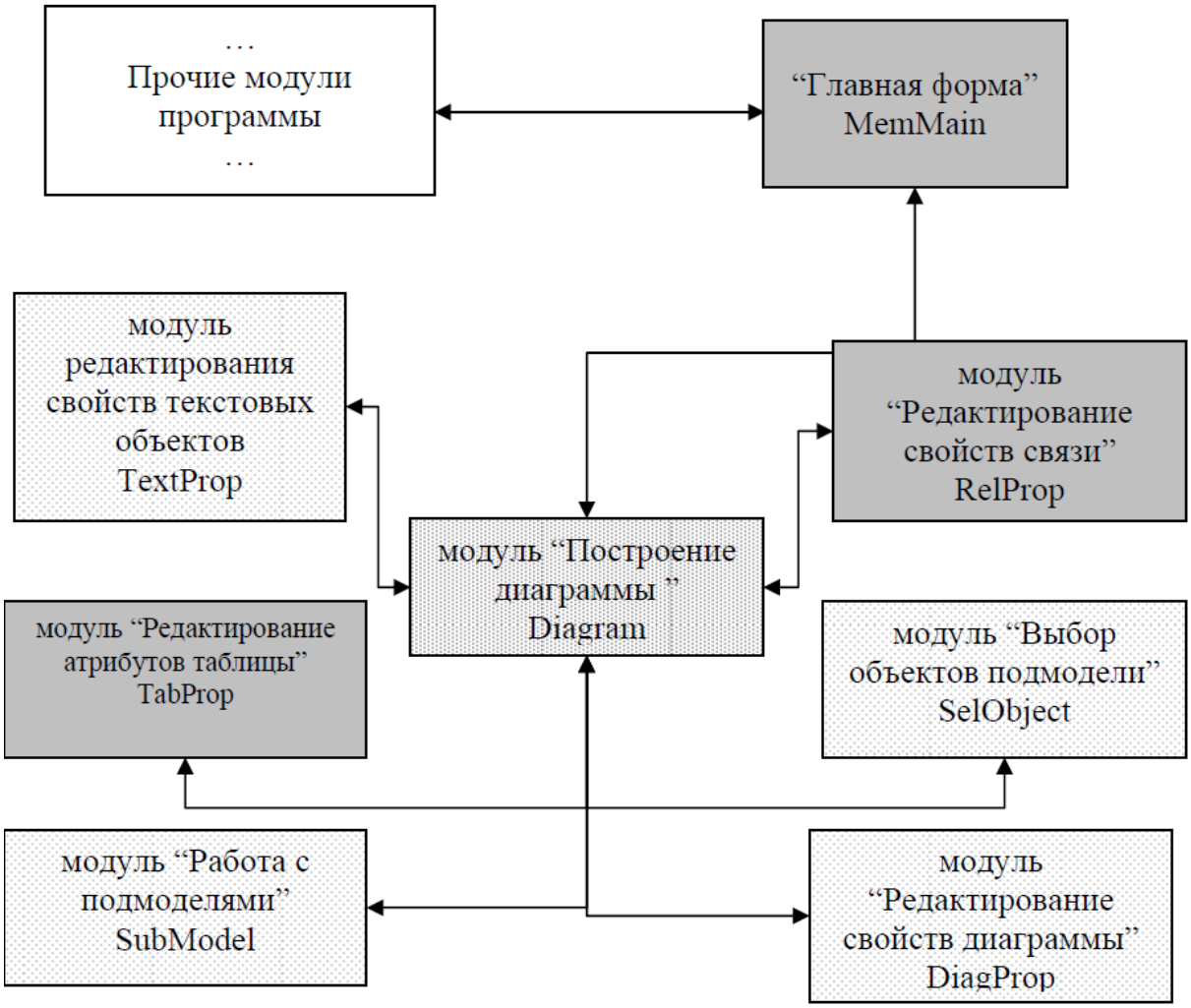


Рис. 4.12. Упрощенная модульная структура программы

4.5.2. Структура представления отношений в памяти программы

Программа дополнительно использует модуль StrDef в котором определена структура для хранения информации в дереве главной формы программы.

```
type
  PExplorerNode = ^TExplorerNode;
  TExplorerNode = record
    ConnectString: string;
    ServerVersion:integer;
    NodeType: integer;
    ObjectID: integer;
    ObjectName: string;
```

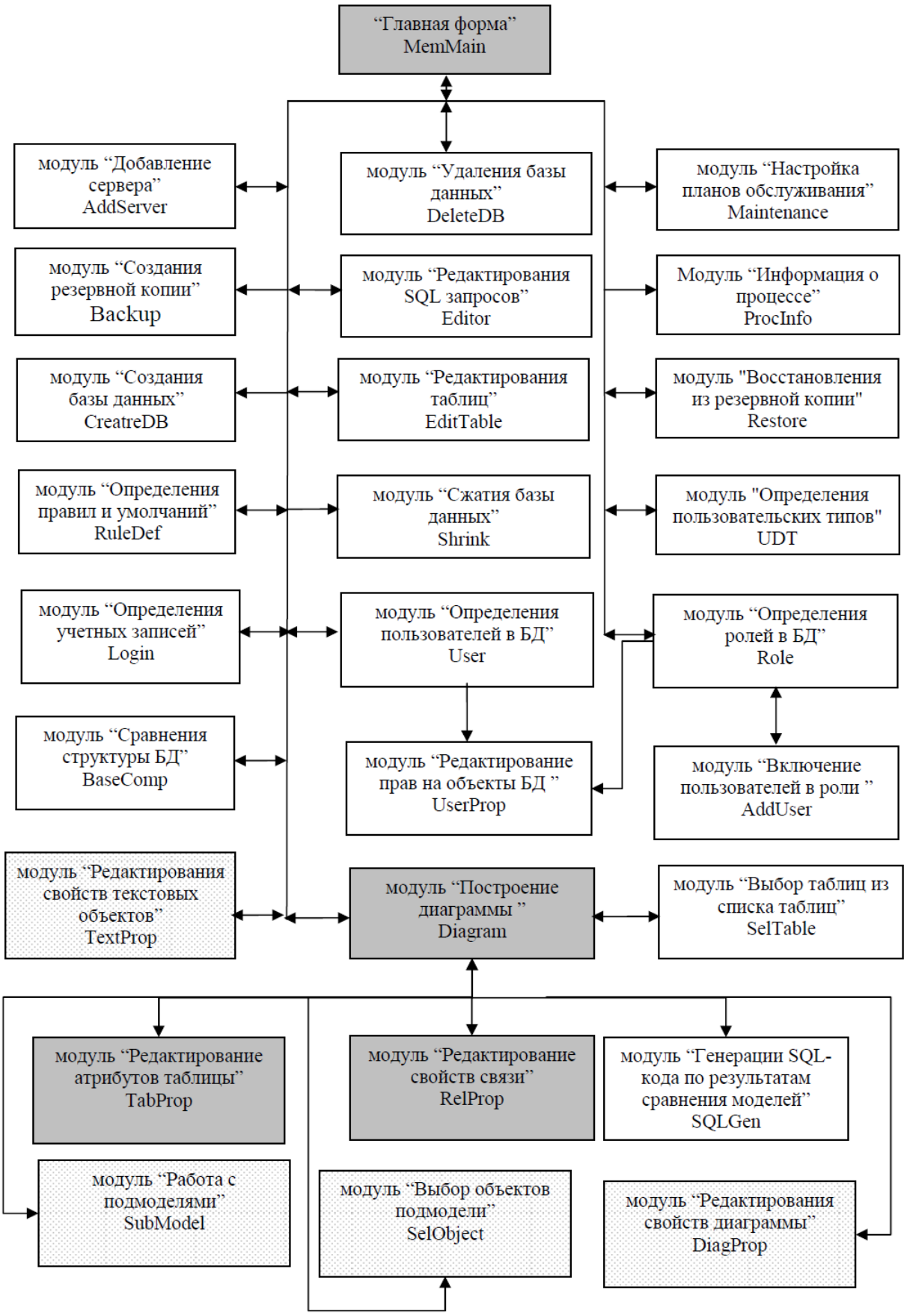


Рис. 4.13. Модульная структура программы

```
    IsPopulate: integer;
    {NodeType соответствует:
     0 - Корневой узел сервера
     1 - Базы данных
     2 - Подключения
     3 - Блокировки
     4 - Планы обслуживания
     5 – Учетные записи
     11 - БД
     21 - список таблиц
     22 - список просмотров
     23 - Хранимые процедуры
     24 - Функции
     25 - Пользовательские типы данных
     26 - Правила
     27 - Умолчания
     50 - Безопасность
     51 - Пользователи
     52 - Роли
     53 - Схемы
    }
  end;
```

Определены структуры для хранения ER-диаграмм (рис. 4.14). В целом структуры для хранения данных повторяют отношения, используемые программой в служебной схеме данных для хранения ER-диаграмм в базе данных. Однако существуют определенные отличия, связанные со способом хранения данных – в памяти и на диске. Стрелками на рисунке обозначены указатели на соответствующие типы данных.

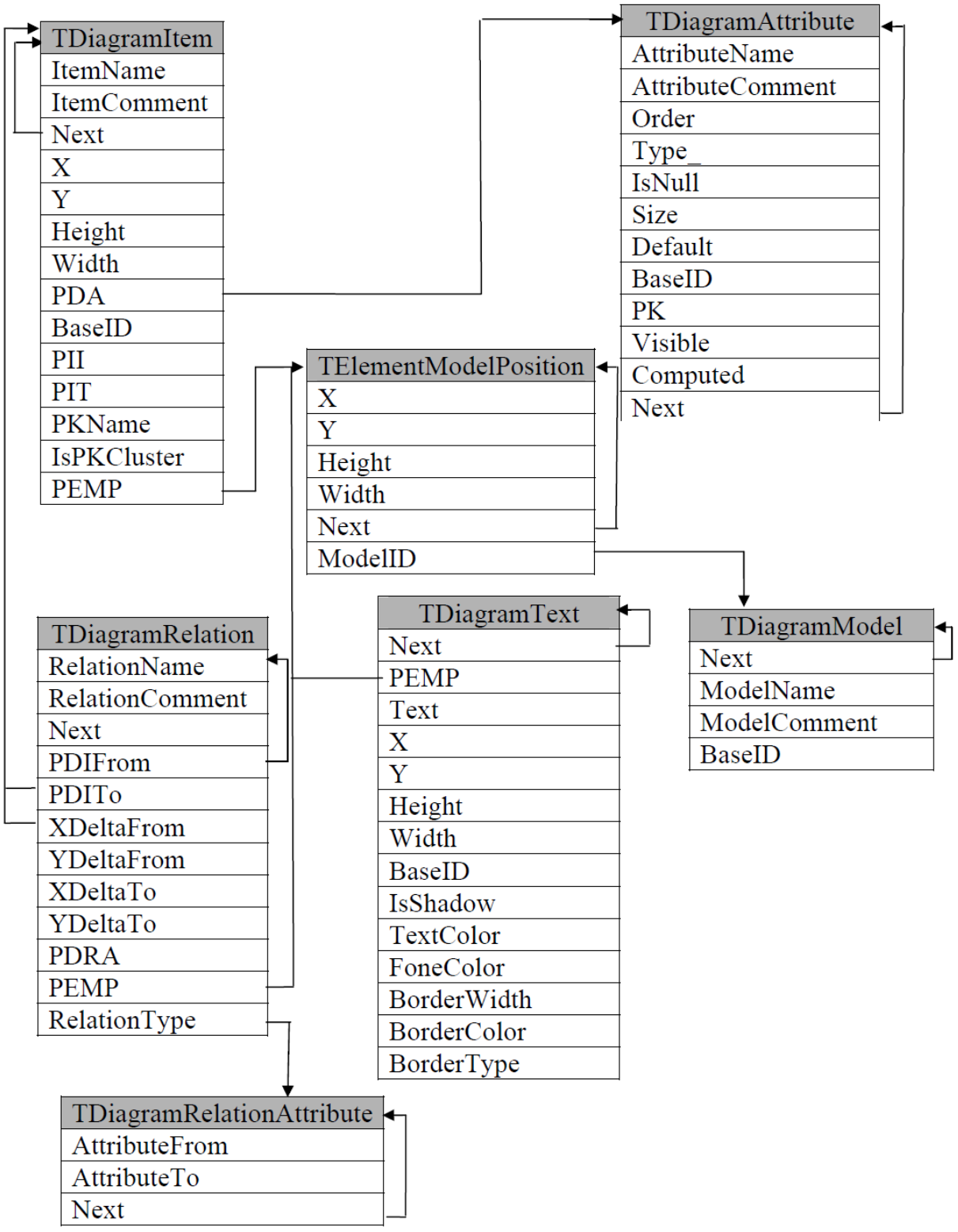


Рис. 4.14. Структуры для хранения диаграмм

Отсутствует структура, соответствующая отношению MEMDiagrams. Вместо этого окно ER-диаграммы использует два указателя – GPDI и

GPDR – указатели на списки структур TDiagramItem и TDiagramRelation соответственно. В первом списке хранятся отношения, которые используются в диаграмме. Во втором списке хранятся связи между отношениями, которые построены в диаграмме. Для каждой связи определен список атрибутов, по которым строится связь, который сохраняется в списке структур типа TDiagramRelationAttribute.

В списке структур типа TDiagramAttribute хранится список атрибутов отношения.

В списках структур типа TElementModelPosition хранится положение элементов на подмодели. Для каждого элемента, который может быть помещен на подмодель (TDiagramItem, TDiagramRelation, TDiagramText) может быть определено его место на главной подмодели, которая всегда присутствует в диаграмме и на ней доступны все объекты, используемые в модели. Данные о расположении элементов на главной подмодели сохраняются в соответствующих полях структур. Для TDiagramItem и TDiagramText это поля X,Y, Height, Width, а для объектов TDiagramRelation это поля XDeltaFrom, YDeltaFrom и XDeltaTo, YDeltaTo. Данные о расположении объектов на подмоделях хранятся в полях X,Y, Height и Width структуры TElementModelPosition. В поле ModelID хранится идентификатор, используемые для идентификации подмодели. Для подмоделей, не сохраненных в базу, для идентификации используются временные отрицательные номера.

Подмодели сохраняются в структуре TDiagramModel. В поле ModelName содержится название подмодели, в поле ModelComment – произвольный текстовый комментарий.

4.6. Перспективы исследования

4.6.1. Многопоточное ведение журнала

Строгая гарантия долговечности транзакций в традиционной конструкции СУБД сериализует транзакции при фиксации журнала (т.е. LSN), и диспетчер журналов становится узким местом производительности при обработке транзакций. Поскольку менеджер журналов в SO-СУБД регулирует скорость ведения журнала во внутреннем хранилище, пропускная способность восстановления страниц во внутреннем хранилище может быть ограничена менеджером журналов. Однако существующие исследования предлагают быстрый, масштабируемый дизайн ведения журнала для максимальной пропускной способности транзакций без ущерба для корректности [4.4]. В сочетании с менеджером продвинутого дизайна, менеджер журналов может легко управлять внутренним хранилищем, организованным с помощью классификатора одной страницы в одном потоке журналов. Осуществимый подход заключается в реализации внутреннего хранилища с несколькими потоками журналов, но существующим ведением журнала в стиле ARIES невозможно реализовать правильное многопоточное ведение журнала.

4.6.2. Восстановление страницы балансировки нагрузки

SO-СУБД использует асинхронную конвейерную архитектуру для параллельного восстановления страниц. Поскольку частая остановка конвейера может повлиять на производительность обработки транзакций в интерфейсе, даже назначение задания восстановления страницы или балансировки нагрузки имеет важное значение для предотвращения остановки конвейера. Изучено два подхода к параллельному восстановлению страниц: назначение потоков для работы и назначение потоков для данных. Текущая реализация подхода "поток-данные" может

недостаточно эффективно использоваться при асимметрии или изменении рабочей нагрузки, и это приводит к остановке конвейера. Необходимо учитывать динамическое или адаптивное назначение задания по восстановлению страницы для асимметрии рабочей нагрузки или изменения рабочей нагрузки.

4.7. Выводы

Высокая доступность облачных СУБД часто достигается за счет отделения компонентов СУБД от физического хранилища. Однако в этой несвязанной организации синхронное ведение журнала транзакций в обычной СУБД могут снизить производительность обработки транзакций. Кроме того, очистка страниц по сети потребляет значительную пропускную способность сети и часто мешает ведению журнала базы данных в общей пропускной способности сети. И поэтому это невозможно для высокопроизводительных систем обработки транзакций.

Предложена организация распределенной СУБД, которая состоит из внешнего сервера для обработки транзакций и внутреннего сервера для обеспечения надежности данных при проектировании разъединения вычислений и данных. Обоснование дизайна для несвязанной организации может быть сведено к трем позициям: (1) модифицированная подсистема ведения журнала для повышения производительности обработки транзакций даже при более длительном завершении ввода-вывода с задержкой в сети, (2) сокращение сетевого трафика за счет очистки только журналов транзакций и (3) внешний процесс, упреждающий восстановление страниц для устойчивости системы при минимальных системных ресурсах. Реализован и исследован прототип дизайна поверх реальных СУБД. Экспериментальные результаты показали, что прототип может достичь на 89% большей пропускной способности, чем базовая СУБД с рабочей нагрузкой, измеренной в sysbench.

Представлена сервис-ориентированная архитектура несвязанной СУБД, отличающаяся модифицированной подсистемой ведения журнала транзакций и механизмом его очистки с нулевой памятью, обеспечивающая снижение служебного сетевого трафика и повышение безопасности обработки при проектировании разделения вычислений и данных.

Список источников к главе 4

4.1. DeWitt, D.J., Katz, R.H., Olken, F., Shapiro, L.D., Stonebraker, M.R., Wood, D.A.: Implementation techniques for main memory database systems. ACM 14, 1–8 (1984)

4.2. Kopytov, A.: Sysbench: a system performance benchmark. http://sysbench.sourceforge.net (2004)

4.3. Tpcc-mysql. https://github.com/Percona-Lab/tpcc-mysql

4.4. Jung, H., Han, H., Kang, S.: Scalable database logging for multicores. Proc. VLDB Endow. 11(2), 135–148 (2017)

Заключение

Цель работы - разработка математического и программного обеспечения безопасного управления репликациями в масштабируемых СУБД на основе мультиагентного самонастраивающегося алгоритма.

В процессе выполнения диссертационного исследования получены следующие основные результаты:

1. Создана модель частных облачных вычислительных систем с подсистемами хранения данных на основе NoSQL на обобщенных стохастических сетей Петри, обеспечивающая корректную оценку пропускной способности системы при различных рабочих нагрузках, доступности и простоях

2. Разработан алгоритм динамической балансировки системы, обеспечивающий повышение производительности и QoS с одновременным определением приоритетов процессов.

3. Разработан мультиагентный самонастраивающийся алгоритм репликации, обеспечивающий конфигурирование гибридного промежуточного программного обеспечения репликации без перезапуска СУБД в режиме реального времени.

4. Создана сервис-ориентированная архитектура СУБД, обеспечивающая снижение служебного сетевого трафика и повышение безопасности обработки при проектировании разделения вычислений и данных.

5. Представлена архитектура человеко-машинной системы проектирования базовой структуры БД, обеспечивающая автоматизацию динамического включения новых подмоделей в структуру БД.

6. Элементы программного обеспечения зарегистрированы в ФИПС.

Рекомендации и перспективы дальнейшей разработки темы

1. Результаты диссертационного исследования рекомендуются к

применению в больших информационных системах, основанных на облачной обработке данных, для повышения скорости и безопасности обработки данных.

2. Дальнейшая разработка темы будет направлена на практическую реализацию теоретических и алгоритмических результатов, интеграцию в наиболее распространенные системы облачных вычислений и облачные хранилища. Безотносительно облачных терминов, развитие результатов будет направлено на улучшение безопасности репликаций, повышение эффективности алгоритмов динамической балансировки системы.

Список литературы

1. Азиз А.Е. Автоконфигурируемая репликация СУБД со специальным обучением для повышения безопасности// Системы управления и информационные технологии, №4(86), 2021. – С. 50-55.

2. Азиз А.И. Архитектура многосерверной СУБД с раздельной обработкой транзакций и распределением вычислений// Системы управления и информационные технологии, №3(89), 2022. – С. 66-71

3. Азиз А.И. Механизм обучения автоконфигурируемой репликации СУБД и результаты вычислительного эксперимента// Информационные технологии моделирования и управления, №4(126), 2021. – С. 293-307

4. Барабанова И.А., Кравец О.Я. Анализ модификации данных в многоуровневых клиент-серверных системах с отказами// Прикладная математика и информатика: современные исследования в области естественных и технических наук. Матер. IV научно-практической международной конференции (школы-семинара) молодых ученых: в 2 частях. Тольятти, 2018. С. 31-35.

5. Болгов В.С. Алгоритм взаимосвязи приложения-клиента с удаленным сервером// Современные проблемы информатизации в моделировании и социальных технологиях: Сб. трудов. Вып. 16. - Воронеж: Изд-во "Научная книга", 2011. – С. 209-210.

6. Бондаренко Ю.В., Азиз А.Е. Разработка алгоритма распределения ресурсов в распределенных системах на основе двухкритериальной оценки процессов// Моделирование, оптимизация и информационные технологии. 2021; 9(3). https://moitvivt.ru/ru/journal/pdf?id=1025.

7. Бондаренко Ю.В., Азиз А.И. Проблемы оптимизации и самозащиты репликаций при размещении заданий в облачных вычислениях // Труды Всероссийской научной конференции «Достижения науки и технологий-ДНиТ-2021». – Красноярск, 2021. http://ru-

conf.domnit.ru/media/filer_public/75/1f/751ff3d0-e0e8-46ad-9efd-ba7c39fdb42d/3005-dnit-2021.pdf.

8. Бондаренко Ю.В., Азиз А.И., Бондаренко О.В. Согласованное распределение ресурсов в интегрированных структурах. - Свидетельство о регистрации программы для ЭВМ 2020664335, 11.11.2020. Заявка № 2020663633 от 03.11.2020.

9. Говорский А.Э., Копылов М.В. Особенности сквозного цикла моделирования и проектирования многозвенной клиент-серверной системы// Системы управления и информационные технологии, №1(39), 2010. – С. 27-32

10. Гриневич А.И., Кулямин В.В., Марковцев Д.А., Петренко А.К., Рубанов В.В., Хорошилов А.В. Использование формальных методов для обеспечения соблюдения программных стандартов. – Тр. института системного программирования РАН. - 2006. - Т.10, с. 51-68.

11. Грязнов Н.Г., Димитриев Ю.К., Мелентьев В.А. Оптимизация отказоустойчивого вложения диагностического графа в тороидальные структуры живучих вычислительных систем// Автоматика и телемеханика. 2003. № 4. С. 133-152.

12. Дастин Э., Рэшка Д., Джон П. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. - М.: Лори, 2003.

13. Дейт К. Введение в системы баз данных - 6-е изд. - Киев: Диалектика, 1998. - 784 С.

14. Дейтел Х., Дейтел П., Нието Т. Как программировать для Internet & WWW М.: Бином, 2002, - 1184 С.

15. Денисов А.А., Колесников Д.Н. Теория больших систем управления. Л.: Энергоиздат, 1982. - 288 С.

16. Диго С.М. Проектирование и использование баз данных. М.: Финансы и статистика, 1995. -208 С.

17. Дрожжинов В.И. От теста не уйдешь// Мир ПК, N 2, 1993.

18. Жожикашвили В.А., Вишневский В.М. Сети массового обслуживания. Теория и применение к сетям ЭВМ. М.: Радио и связь, 1988. – 192 С.

19. Журков А.П., Аминев Д.А., Гусева П.А., Мирошниченко С.С., Петросян П.А. Анализ возможностей применения подходов самодиагностирования к распределенной радиотехнической системе наблюдения// Системы управления, связи и безопасности. 2015. № 4. С. 114-122.

20. Задорожный В.Н. Модели и системы. Анализ научного мышления. - Омск, ОМГТУ, 1999. - 100 с.

21. Задорожный В.Н. Статистическое моделирование. - Омск: Изд-во ОмГТУ, 1996. - 92с.

22. Иванов П. Управление информационными системами: базовые концепции и тенденции развития // Открытые системы, №4, 1999, С. 37-43.

23. Камер Д. Компьютерные сети и Internet М.: Вильямс, 2002.- 640 С.

24. Кастаньетто Д. и др. Профессиональное PHP программирование. М.: Символ – Плюс, 2001. - 912 С.

25. Кениг Д., Штоян Д. Методы теории массового обслуживания.- М.: Радио и связь, 1981. – 127 с.

26. Киллелиа П. Тюнинг WEB-сервера. СПб.: Питер, 2003. - 528 С.

27. Клейнен Дж. Статистические методы в имитационном моделировании. - М.: Статистика, 1978. - Вып. 1. - 221 с.; Вып. 2 - 335 с.

28. Клейнрок Л. Вычислительные системы с очередями: Пер. с англ. М.: Мир, 1979 - 600 С.

29. Клейнрок Л. Теория массового обслуживания: Пер. с англ. М.: Машиностроение, 1979 – 432 С.

30. Когаловский М.Р. Энциклопедия технологий баз данных. М.: Финстат, 2002. - 800 С.

31. Кокс Д. Р., Смит У. Л. Теория очередей: Пер. с англ. М.: Мир, 1966 - 218 С.

32. Колесов Ю.Б., Сениченков Ю.Б., Визуальное моделирование, СПб.: Мир и Семья, 2000. - 256 С.

33. Конвей Р., Максвелл В., Миллер Л. Теория расписаний: Пер с англ. / Под ред. Г.П. Башарина. – М.: Наука, 1975 – 159 с.

34. Копылов М.В. Модель двухзвенной архитектуры «клиент – сервер»// Современные проблемы информатизации в моделировании и анализе сложных систем: Сб. трудов. Вып. 12. - Воронеж: "Научная книга", 2007. – С. 177-183.

35. Копылов М.В. Модель трехзвенной архитектуры «клиент – сервер»// Современные проблемы информатизации в проектировании и телекоммуникациях: Сб. трудов. Вып. 12. - Воронеж: "Научная книга", 2007. – С. 352-358.

36. Копылов М.В. Особенности поведения многозвенных клиент-серверных систем на граничных нагрузках// Информационные технологии моделирования и управления. - 2007, №6(40), с. 675-683.

37. Копылов М.В., Говорский А.Э., Солдатов Е.А. Аналитические основы моделирования и проектирования многозвенной клиент-серверной системы// Информационные технологии моделирования и управления, №1(60), 2010. – С. 49-60

38. Кравец О.Я., Барабанова И.А. Обзор способов управления многоуровневыми клиент-серверными системами с отказами// Новый университет. Серия «Технические науки», 12(58), 2016. С. 55-57

39. Кравец О.Я., Барабанова И.А., Соляник С.А. Программа нечёткого к-поиска близких объектов. - Свидетельство о государственной

регистрации программы для ЭВМ №2018666616 от 19.12.2018. М.: ФИПС, 2018.

40. Кравец О.Я., Соляник С.А. Алгоритмизация построения топологии системы управления потоками данных в ячеистых сетях// Системы управления и информационные технологии, №3(73), 2018. С. 66-69.

41. Кравец О.Я., Соляник С.А. Разработка процессной модели управления инновационной деятельностью при разработке наукоемких прикладных IT-решений в условиях импортозамещения в сфере информационных технологий// Информационные технологии моделирования и управления, №2(104), 2017. – С. 115-132.

42. Кравец О.Я., Соляник С.А. Резултат от интегрирането на специален математически и софтуер в практиката на метрологията// Парадигма. Електронно научно списание. Брой 2/2016. - С. 59-74. http://niiparadigma.ru/?page_id=1783.

43. Кравец О.Я., Соляник С.А. Управление вероятностными потоками в беспроводных ячеистых сетях на основе использования ориентированного ациклического графа// Системы управления и информационные технологии, №4(70), 2017. – С. 40-44.

44. Ланг К., Чоу Д. Публикация баз данных в Интернете. – СПб.: Символ-Плюс, 2004.

45. Ларионов А.М., Майоров С.А., Новиков Г.И. Вычислительные комплексы, системы и сети. Л.: Энергоатомиздат, 1987. - 256 С.

46. Лебедев А.Н., Чернявский Е.А. Вероятностные методы в вычислительной технике. М.: Высшая школа, 1986. - 312 С.

47. Левенчук А. Интранет предлагает решения для корпорации// Рынок ценных бумаг, №16, 1996.

48. Липский Н. Комбинаторика для программистов. М.: Мир, 1985. – 374 С.

49. Мазуркевич А., Еловой Д. PHP: настольная книга программиста. М.: Новое знание 2003. -480 С.

50. Майерс Г. Надежность программного обеспечения. - М.: Мир, 1980.

51. Мальцева С.В. Информационное моделирование WEB-ресурсов Интернет. М.: Глобус, 2003. - 216 С.

52. Матвеев В.Ф., Ушаков В.Г. Системы массового обслуживания. М.: Изд-во МГУ, 1984. – 240 С.

53. Мельтцер К., Михальски Б. Разработка CGI-приложений на Perl. М.: Вильямс, 2001. - 400 С.

54. Мехди М., Ронни М. Непрерывное развитие API. Правильные решения в изменчивом технологическом ландшафте. М.: Прогресс книга, 2020.

55. Мещеряков Е.В., Хомоненко А.Д. Публикация баз данных в Интернете Спб.: BHV-Санкт-Петербург, 2001. - 560 С.

56. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. 2-изд. СПб: Питер-пресс, 2002. - 864 С.

57. Павловский Ю. Н. Имитационные модели и системы. М.: ФАЗИС, 2000. - 144 С.

58. Перегудов Ф.П., Тарасенко Ф.П. Введение в системный анализ. М: Высшая школа, 1989. - 367 С.

59. Петренко А., Бритвина Е., Грошев С., Монахов А., Петренко О. Тестирование сетевой инфраструктуры// Открытые системы, 09/2003.

60. Ройс У. Управление проектами по созданию программного обеспечения. - М.: Лори, 2002.

61. Сандберг Б.В. Оптимизация выбора клиент-серверной архитектуры// Всероссийское совещание-семинар “Высокие технологии в региональной информатике”. Тез.докл. Часть 1. Воронеж, Изд-во ВГТУ, 1998. С.76.

62. Сафонов А.И. Методология анализа и проектирования специализированных многозвенных клиент-серверных систем. Воронеж: «Научная книга», 2010.

63. Сафонов А.И., Строгонов В.В. Особенности реализации многозвенной клиент-серверной архитектуры с "доверенным" сервером// Информационные технологии моделирования и управления. - 2005, №4(22), с. 596-599.

64. Свами М., Тхуласираман К. Графы, сети и алгоритмы: Пер. с англ. М.: Мир, 1984. - 455 С.

65. Седова Н.А. Формирование лингвистических переменных для задач судовождения// Эксплуатация морского транспорта. – 2013. – №2(72). – С. 19-23.

66. Седова Н.А., Перечёсов В.С., Седов В.А. Удержание судна на курсе на базе нечеткой логики с учетом скорости судна// Автоматизация процессов управления. – 2013. – № 2. – С.74-79.

67. Слоневская О.С., Вериковский В.А. Проблемы разработки специализированных клиент-серверных приложений на примере разработки информационной системы специализированной клиники// Информационные технологии моделирования и управления, №4(69), 2011. – С. 478-492.

68. Советов Б.Я., Яковлев С.А. Моделирование систем //3-е изд., М:Высшая школа, 2001. - 344 С.

69. Создание Intranet. Официальное руководство Microsoft. - СПб.: BHV, 1998.

70. Соляник А.А. Особенности проектирования технологии мобильных программных агентов в компьютерных сетях на основе “тонких” клиентов// Современные проблемы информатизации в экономике и обеспечении безопасности: Сб. трудов. Вып. 15. - Воронеж: "Научная книга", 2010. – С. 92-95

71. Соляник А.А., Авсеева О.В., Кравец О.Я. Особенности управления процессами конкурентного проектирования программного обеспечения // Системы управления и информационные технологии. 2010. Т. 39. № 1.2. С. 308-312.

72. Соляник А.И., Кравец О.Я. Об аналитическом подходе к метамоделированию механизмов управления региональной социально-экономической системой // Системы управления и информационные технологии. №1.3(31), 2008. – С. 380-384.

73. Соляник А.И., Кравец О.Я. Пути и средства повышения эффективности управления качеством социально-экономической системы // Системы управления и информационные технологии. №2.2(32), 2008. – С. 304-308.

74. Соляник А.И., Кравец О.Я. Системно-структурное моделирование объекта управления на основе проектного подхода // Системы управления и информационные технологии, №4(46), 2011. – С. 43-45.

75. Соляник А.И., Соляник С.А. Формализация программной модели управления территориальной системой с использованием нечетких множеств// Системы управления и информационные технологии, №3(65), 2016. – С. 87-92.

76. Соляник С.А. Графовые инструменты управления пакетами без предварительного оповещения// Информационные технологии моделирования и управления, №3(111), 2018. – С. 234-239.

77. Соляник С.А. Способ определения диагностических способностей системы с полиномиальной оценкой трудоемкости по времени// Информационные технологии моделирования и управления, №6(102), 2016. – С. 437-445.

78. Соляник С.А. Управление потоками данных в беспроводных ячеистых сетях с однократной рассылкой по топологии графа// Системы управления и информационные технологии, №1(71), 2018. – С. 85-88.

79. Соляник С.А., Кравец О.Я. Исследование устойчивости процесса синтеза статических программных систем с блокировками// Экономика и менеджмент систем управления, №3.1(25), 2017. – С. 166-179.

80. Соляник С.А., Кравец О.Я. Модели и алгоритмы оперативного управления вероятностными процессами в многоуровневой специализированной системе// Информационные технологии моделирования и управления, №4(100), 2016. – С. 70-80.

81. Соляник С.А., Кравец О.Я. Размита моделиращи дейности лаборатории за калибриране като структурна предмет на териториално система, гарантираща единство на измеренията // Парадигма. Електронно научно списание. Брой 2/2016. - С. 145-158. http://niiparadigma.ru/?page_id=1783.

82. Соляник С.А., Кравец О.Я. Разработка процессной модели создания прикладных IT-решений для задач управления// Экономика и менеджмент систем управления, №3(25), 2017. – С. 79-86.

83. Соляник С.А., Кравец О.Я. Элементы синтеза статических программных систем с компонентами параллелизма// Информационные технологии моделирования и управления, №4(106), 2017. – С. 274-284.

84. Соляник С.А., Соляник А.И. Рационализация плановой диагностики дискретно-событийных систем в «холодном» режиме// Системы управления и информационные технологии, №4.1(66), 2016. – С. 184-188.

85. Стивен Р. Программирование в Win32 API на Visual Basic. М.: ДМК Пресс, 2017.

86. Стрелкова Е. Интеграция данных предприятия / Открытые системы, №4, 2003, С. 58-60.

87. Технология системного моделирования/ Е.Ф. Аврамчук, А.А. Вавилов, С.В. Емельянов и др. / Под ред. С.В. Емельянова. – М.: Машиностроение, 1988. – 320 с.

88. Томашевский В.Н., Жданова Е.Г. Имитационное моделирование в среде GPSS. Серия "Факультет". М.: Бестселлер, 2003. 400 С.

89. Томсетт Р. Радикальное управление ИТ проектами. М.: Лори, 2005.

90. Увайсов С.У., Иванов И.А., Кошелев Н.А. Методика обеспечения диагностируемости электронных средств космических аппаратов по ранговому критерию на ранних этапах проектирования// Качество. Инновации. Образование. 2012. № 1 (80). С. 60-63.

91. Уемов А.И. Системный подход и общая теория систем. М.: Мысль, 1978.- 272 С.

92. Управление качеством продукции. Введение в системы менеджмента качества / С. Пономарев, С. Мищенко, В.Я. Белобрагин. М.: Стандарты и качество, 2005.

93. Феллер В. Введение в теорию вероятностей и её приложения: в 2-х т. - М.: Мир,1967. - т.1, 498 с.

94. Феррари Д. Оценка производительности вычислительных систем. М.: Мир, 1981. - 576 С.

95. Французов Д. Оценка производительности вычислительных систем. Открытые системы, №2, 1996, С. 58-66.

96. Фролов А.В. Фролов Г.В. Локальные сети персональных компьютеров. М.: ДИАЛОГ-МИФИ, 1993. - 176 С.

97. Харари Ф. Теория графов. М.: Мир, 1973. - 300 С.

98. Харрингтон Дж. Проектирование реляционных баз данных. М: Лори-Пресс, 2000. - 230 С.

99. Харт Д.М. Системное программирование в среде Win32 - 2-е издание. М.: Вильямс, 2001. 464 С.

100. Хилайер С., Мизик Д. Программирование Active Server Pages. М.: Русская редакция. 2000. - 320 С.

101. Храмцов П.Б., Брик С.А. и др. Основы web-технологий. М.: ИНТУИТ.ру, 2003. – 512 С.

102. Цаленко М.Ш. Моделирование семантики в базах данных. М.: Наука, 1989. - 287 С.

103. Черепухин А.Н. Аналитическое моделирование многозвенной архитектуры клиент-серверных систем// Вестник Воронежского государственного технического университета. Т. 1, №5, 2005. С.8-13.

104. Черепухин А.Н. Модели многозвенной архитектуры клиент-серверных систем// Современные сложные системы управления. Сб. тр. НПК. Т. 2. – Воронеж: ВГАСУ, 2005. – С. 20-28.

105. Черепухин А.Н. Программный комплекс "Амбулаторная поликлиника" на основе клиент-серверных технологий обработки распределенных данных// Врач-аспирант. – 2006, №5(14), с. 473-481.

106. Черняк Л. Intranet - объективная реальность, данная нам// Рынок ценных бумаг, №3, 1997.

107. Чжо З.Е., Чжо З.Л., Наинг Л.А. Разработка методики децентрализованной диагностики и диагностируемости с использованием модели тестирования// Вести высших учебных заведений Черноземья. 2015. № 3. С. 60-70.

108. Abramova V., Bernardino J. NoSQL databases: MongoDB vs Cassandra// International C* Conference on Computer Science and Software Engineering, ser. C3S2E ’13. ACM, 2013, pp. 14-22.

109. Agrawal S., Chaudhuri S., Narasayya V.R. Automated selection of materialized views and indexes in SQL databases// VLDB, 2000, pp. 496-505.

110. Algorithm and model for improve the avoiding of deadlock with increasing efficiency of resource allocation in cloud environment/ Yu. V.

Bondarenko, A.E. Azeez// Journal of Physics: Conference Series. 2021, 1902(1). - P. 012054.

111. Algorithm for CPU resource allocator case study and compering between ordinary and ML Algorithms / A.E. Azeez, Yu. V. Bondarenko // IOP Conf. Ser.: Mater. Sci. Eng., 2020, 928(3). P. 032064.

112. http://bookkeeper.apache.org/distributedlog/.

113. Azeez A.E. Research on designing and optimizing the reliability of transaction processing in a service-oriented DBMS// Modern informatization problems in economics and safety (MIP-2022'ES): Proceedings of the XXVII-th International Open Science Conference (Yelm, WA, USA, January 2022). - Yelm, WA, USA: Science Book Publishing House, 2022. – p. 20-40

114. Azeez A.E., Bondarenko Yu.V. Algorithm and model for improve the avoiding of deadlock with increasing efficiency of resource allocation in cloud environment by using process time execution// Актуальные проблемы прикладной математики, информатики и механики: сб. тр. Междунар. науч. конф. - Воронеж: ВГУ, 2021. С. 493-498.

115. Balbo G. Introduction to Stochastic Petri Nets. Springer Berlin Heidelberg, pp. 84 - 155, 2001.

116. Bondarenko Ju.V., Azeez A.I. Algorithm for improving the resource allocation depending on the jobs request criteria// Актуальные проблемы прикладной математики, информатики и механики: сб. тр. Междунар. науч. конф. - Воронеж: ВГУ, 2022. С. 1697-1700.

117. Bruneo D. A stochastic model to investigate data center performance and QoS in IaaS cloud computing systems// IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 560-569, 2014.

118. Campbell, D.: Service oriented database architecture: App serverlite? In: Proceedings of the 2005 ACM SIGMOD international conference on Management of Data, pp. 857–862. ACM (2005)

119. Castillo G., Rouskas N., Harfoush K. Efficient QoS resource

management for heterogeneous Grids. 22nd. IEEE International Parallel and Distributed Processing Symposium (IPDPS'08), Miami, Florida, US. 2008;1-15.

120. Cattell R. Scalable SQL and noSQL data stores// SIGMOD Rec., vol. 39, no. 4, pp. 12-27, 2011.

121. Chakraborty S., Yeh C. H. A Simulation Based Comparative Study of Normalization Procedures in Maldistributed Decision Making. Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases.2007; 102–109.

122. Chalkiadaki M., Magoutis K. Managing service performance in noSQL distributed storage systems// 7th Workshop on Middleware for Next Generation Internet Computing, ser. MW4NG ’12. ACM, 2012.

123. Chen L., Xu Z., Wang H., Liu S. An ordered clustering algorithm based on K-means and the PROMETHEE method. International Journal of Machine Learning and Cybernetics. 2018; 9(6):917-926.

124. Coburn, J., Bunker, T., Schwarz, M., Gupta, R., Swanson, S.: From aries to mars: Transaction support for next-generation, solid-state drives. In: Proceedings of the twenty-fourth ACM symposium on operating systems principles, pp. 197–212. ACM (2013)

125. Cooper B.F. Yahoo! cloud system benchmark (YCSB). https://github.com/brianfrankcooper/YCSB, 2018.

126. Curino C., Jones E., Zhang Y., Madden S. Schism: a workload-driven approach to database replication and partitioning// Proc. VLDB Endow., 2010, vol. 3(1-2), pp. 48-57.

127. Davoudian A., Chen L., Liu M. A survey on noSQL stores// ACM Comput. Surv., vol. 51, no. 2, pp. 40:1-40:43, 2018.

128. de Lima Q.V. et al. Performability evaluation of emergency call center// Performance Evaluation, vol. 80, pp. 27 - 42, 2014.

129. Dede E. et al. Performance evaluation of a mongoDB and hadoop platform for scientific data analysis// 4th ACM Workshop on Scientific Cloud Computing, ser. Science Cloud ’13. ACM, 2013.

130. Deka G.C. A survey of cloud database systems// IT Professional, vol. 16, no. 2, pp. 50-57, 2014.

131. DeWitt, D.J., Katz, R.H., Olken, F., Shapiro, L.D., Stonebraker, M.R., Wood, D.A.: Implementation techniques for main memory database systems. ACM 14, 1–8 (1984)

132. Diaconu C., Freedman C., Ismert, E., Larson P.A., Mittal P., Stonecipher R., Verma N., Zwilling, M.: Hekaton: Sql server's memory-optimized oltp engine. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 1243–1254. ACM (2013)

133. Dias K., Ramacher M., Shaft U., Venkataramani V., Wood G. Automatic performance diagnosis and tuning in Oracle// CIDR, 2005, pp. 84-94.

134. Dipietro S., Casale G., Serazzi G. A queuing network model for performance prediction of Apache Cassandra// 10th EAI International Conference on Performance Evaluation Methodologies and Tools on 10th EAI International Conference on Performance Evaluation Methodologies and Tools. ICST (Institute for Computer Sciences, Social Informatics and Telecommunications Engineering), 2017.

135. Duan S., Thummala V., Babu S. Tuning database configuration parameters with ituned// Proc. VLDB Endow., 2009, vol. 2(1), pp. 1246-1257.

136. French C.D. One size fits all" database architectures do not work for DSS// Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, pp. 449-450.

137. Gandini A. et al. Performance evaluation of noSQL databases// Computer Performance Engineering. Springer International Publishing, 2014.

138. Garcıa J., Fernandez F. A comprehensive survey on safe reinforcement learning// J. Mach. Learn. Res., 2015, vol. 16(1), pp. 1437-1480.

139. George L. HBase: The Definitive Guide: Random Access to your Planet-Size Data. - Sebastopol: O'Reilly Media Inc., 2011.

140. Ghosh R. et al. End-to-end performability analysis for infrastructure-

as-a-service cloud: An interacting stochastic models approach// 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, 2010.

141. Gomes C., Tavares E.A.G., de Nogueira M.O. Jr. Energy consumption evaluation of noSQL DBmss// 15 WPerformance - Workshop em Desempenho de Sistemas Computacionais e de Comunicação, 2016.

142. Graefe G., Guy W., Sauer C. Instant recovery with write-ahead logging: page repair, system restart, and media restore. Synth. Lect. Data Manag. 6(5), 1–85 (2014)

143. Gudivada V.N., Rao D., Raghavan V.V. NoSQL systems for big data management// 2014 IEEE World Congress on Services, 2014.

144. Harder, T. Dbms architecture - the layer model and its evolution. Datenbank-Spektrum 13, 45–57 (2005)

145. Hintjens P. ZeroMQ: Messaging for many Applications. Sebastopol: O'Reilly Media Inc., 2013.

146. Irmert F., Daum M., Meyer-Wegener K.: A new approach to modular database systems. In: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management, pp. 40–44. ACM (2008)

147. Jiang H., Ni T. PB-FCFS-a task scheduling algorithm based on FCFS and backfilling strategy for grid computing. Proceedings of Joint Conferences on Pervasive Computing (JCPC). 2009; 507- 510.

148. Jung H., Han H., Kang S.: Scalable database logging for multicores. Proc. VLDB Endow. 11(2), 135–148 (2017)

149. Kirsal Y. et al. Analytical modeling and performability analysis for cloud computing using queuing system// in 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), 2015.

150. Klein J. et al. Performance evaluation of noSQL databases: A case study// 1st Workshop on Performance Analysis of Big Data Systems, ser. PABS '15. ACM, 2015.

151. Kopytov A. Sysbench: a system performance benchmark. http://

sysbench.sourceforge.net (2004)

152. Li G., Zhou X., Li S., Gao B. Qtune: a query-aware database tuning system with deep reinforcement learning// Proc. VLDB Endow, 2019, vol. 12(12), pp. 2118-2130.

153. Malviya N., Weisberg A., Madden S., Stonebraker M.: Rethinking main memory OLTP recovery. In: Proceedings of the 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 604–615. IEEE (2014)

154. Marcus R., Papaemmanouil O. Deep reinforcement learning for join order enumeration// Proc. of the First Int. Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018, pp. 3:1-3:4.

155. Mareschal B., Brans Jean-Pierre. PROMETHEE methods. International Series in Operations Research and Management Science.2014;78(2):163-195.

156. Marsan M. et al. An introduction to generalized stochastic Petri Nets// Microelectronics Reliability, vol. 31, no. 4, pp. 699 - 725, 1991.

157. Mohan C., Haderle D., Lindsay B., Pirahesh H., Schwarz P.: Aries: a transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst. 17(1), 94–162 (1992)

158. Montgomery D. Design and Analysis of Experiments. John Wiley & Sons, 2008.

159. Morff A.R., Paz D.R., Hing M.M., Gonzalez L.M.G. A reinforcement learning solution for allocating replicated fragments in a distributed database// Computacion y Sistemas, 2007, vol. 11(2), pp. 117-128.

160. Murata T. Petri nets: Properties, analysis and applications// Proceedings of the IEEE, vol. 77, no. 4, pp. 541-580, 1989.

161. Openstack cloud software. https://www.openstack.org/software/, 2018.

162. Osman R., Knottenbelt W.J. Database system performance evaluation models: A survey// Performance Evaluation, vol. 69, no. 10, pp. 471 - 493, 2012.

163. Osman R., Piazzolla P. Modeling replication in noSQL datastores// Quantitative Evaluation of Systems. Springer International Publishing, 2014.

164. Pandis I., Johnson R., Hardavellas N., Ailamaki A. Data-oriented transaction execution. Proc. VLDB Endow. 3(1–2), 928–939 (2010)

165. Pandis I., Tozun P., Johnson R., Ailamaki A. Plp: page latchfree shared-everything oltp. Proc. VLDB Endow. 4(10), 610–621 (2011)

166. Pavlo A., Curino C., Zdonik, S. Skew-aware automatic database partitioning in shared-nothing, parallel OLTP systems // Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 61–72. ACM (2012)

167. Powell D. Delta-4: A Generic Architecture for Dependable Distributed Computing, vol. 1. Springer, Cham, 2012.

168. Puterman M.L. Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, Hoboken, 2014.

169. Reid C., Bernstein, P., Wu M., Yuan X. Optimistic concurrency control by melding trees. //Proc. VLDB Endow. 4(11), 944–955 (2011)

170. Schiefer K.B., Valentin G. Db2 universal database performance tuning// IEEE Data Eng. Bull., 1999, vol. 22(2), pp. 12-19.

171. Schnaitter K., Abiteboul S., Milo T., Polyzotis N. Colt: continuous on-line tuning// Proc. of the 2006 ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 793-795.

172. Silberschatz A., P. Baer Galvin, Gagne G. Operating Systems Concepts. New York: John_Wiley_&_Sons. 2008;(8):741-750.

173. Silva B. et al. Astro: An integrated environment for dependability and sustainability evaluation// Sustainable Computing: Informatics and Systems, vol. 3, no. 1, pp. 1 - 17, 2013.

174. Stonebraker M., Rowe L.A. The Design of Postgres, vol. 15. ACM, New York, 1986.

175. Stonebraker M., Madden S., Abadi D.J., Harizopoulos S., Hachem N., Helland P. The end of an architectural era: (it's time for a complete rewrite). // Proceedings of the 33rd International Conference on Very Large Data Bases, pp. 1150–1160. VLDB Endowment (2007)

176. Stonebraker M., Weisberg A. The voltdb main memory dbms. IEEE Data Eng. Bull. 36(2), 21–27 (2013)

177. Storm A.J., Garcia-Arellano C., Lightstone S.S., Diao Y., Surendra M. Adaptive self-tuning memory in db2// Proc. of the 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1081-1092.

178. Subasu I., Ziegler P., Dittrich K.R. Towards service-based data management systems. In: Proceedings of the Workshop on Datenbanksysteme in Business, Technologie und Web (BTW 2007), pp. 3–86,130. Citeseer (2007)

179. Sullivan D.G., Seltzer M.I., Pfeffer A. Using Probabilistic Reasoning to Automate Software Tuning, vol. 32. ACM, New York, 2004.

180. Sutton R.S., Barto A.G. Reinforcement Learning: An Introduction. MIT Press, Cambridge, 2018.

181. Tang E., Fan Y. Performance comparison between five noSQL databases// 2016 7th International Conference on Cloud Computing and Big Data (CCBD), pp. 105-109, 2016.

182. Thamsen L, Verbitskiy I, Beilharz J, Renner T, Polze A, Kao O. Ellis: Dynamically Scaling Distributed Dataflows to Meet Runtime Targets. Proc. Int/ Conf. Cloud Comput Technol Sci CloudCom. 2017;(37)146–153.

183. Thamsen L., Renner T., Kao O. Continuously Improving the Resource Utilization of Iterative Parallel Dataflows. Proceedings of the 6th International Workshop on Big Data and Cloud Performance, ser. DCPerf 2016. IEEE. 2016;1(4):1–6.

184. Tok W.H., Bressan, S. Dbnet: A service-oriented database

architecture. In: Proceedings of the 17th International Workshop on Database and Expert Systems Applications, DEXA'06. pp. 727–731. IEEE (2006)

185. Tpcc-mysql. https://github.com/Percona-Lab/tpcc-mysql

186. Triantaphyllou E. Multi-Criteria Decision-Making Methods in Multi-Criteria Decision-Making Methods, a Comparative Study. Applied Optimization. 2000; 44(1): 5–21.

187. Trivedi K.S., Hunter S., Garg S., Fricks R. Reliability analysis techniques explored through a communication network example. North Carolina State University. Series/Report No.: TR-96/32. 1996.

188. Trummer I., Moseley S., Maram D., Jo S., Antonakakis J. Skinnerdb: regretbounded query evaluation via reinforcement learning// Proc. VLDB Endow., 2018, vol. 11(12), pp. 2074-2077.

189. Valentin G., Zuliani M., Zilio D.C., Lohman G., Skelley A. Db2 advisor: an optimizer smart enough to recommend its own indexes// Proc. of 16th Int. Conf. on Data Engineering (Cat. no. 00CB37073), 2000, pp. 101-110.

190. Ventura L., Antunes N. Experimental assessment of noSQL databases dependability// 2016 12th European Dependable Computing Conference (EDCC), 2016.

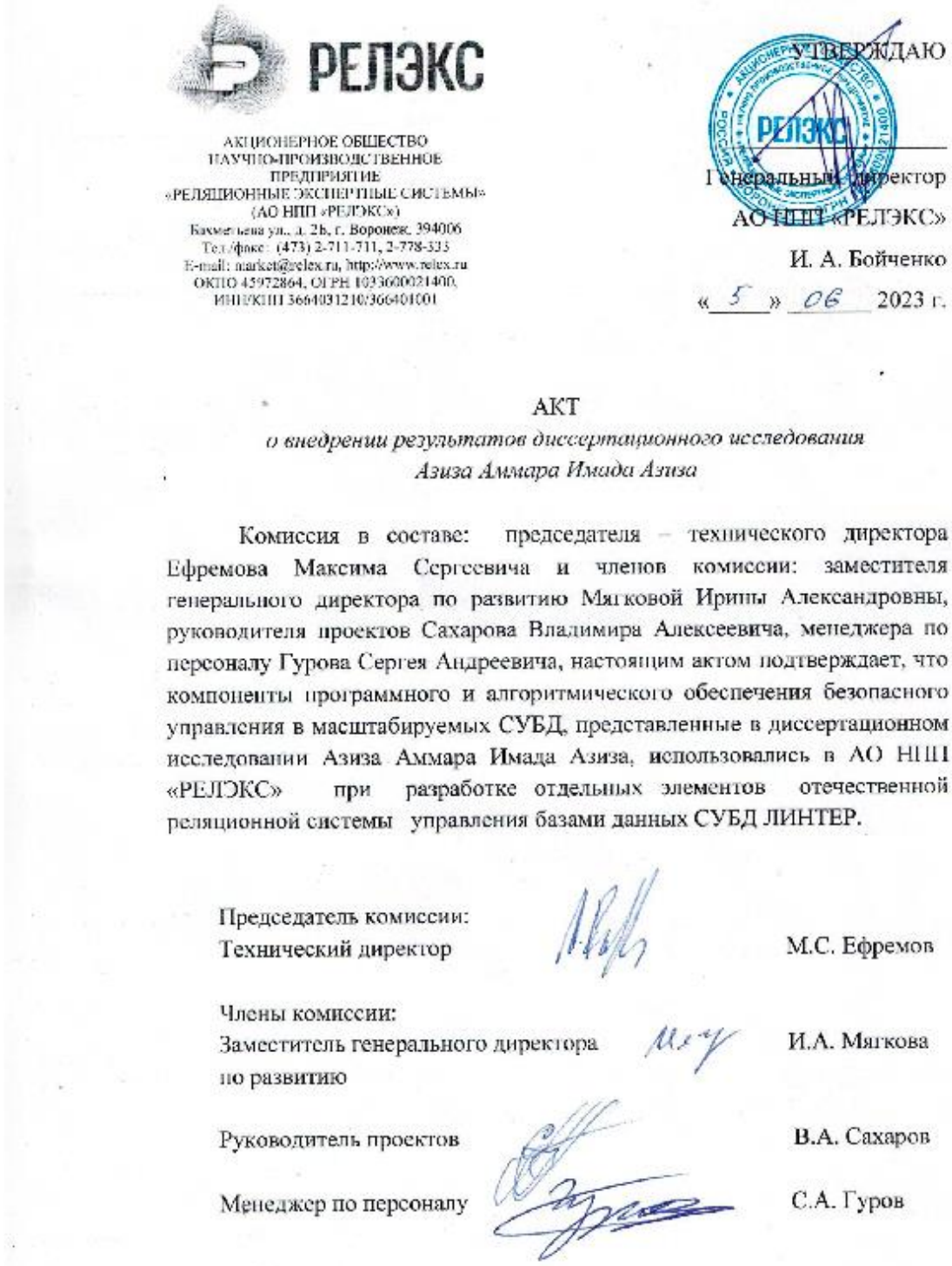
191. Watkins C.J., Dayan P. Q-learning// Mach. Learn., 1992, vol. 8(3-4), pp. 279-292.

192. Wiesmann M., Pedone F., Schiper A., Kemme B., Alonso G. Understanding replication in databases and distributed systems// Proc. 20th IEEE Int. Conf. on Distributed Computing Systems, 2000, pp. 464-474.

193. Yu L., Chen L., Cai Z., Shen H., Liang Y., Pan Y. Stochastic Load Balancing for Virtual Resource Management in Datacenters. IEEE Transactions on Cloud Computing. 2018; 8(2):459–472.

194. Zhang J. An end-to-end automatic cloud database tuning system using deep reinforcement learning// Proc. of the 2019 Int. Conf. on Management of Data, 2019, pp. 415-432.

Приложения



РЕЛЭКС

АКЦИОНЕРНОЕ ОБЩЕСТВО
НАУЧНО-ПРОИЗВОДСТВЕННОЕ
ПРЕДПРИЯТИЕ
«РЕЛЯЦИОННЫЕ ЭКСПЕРТНЫЕ СИСТЕМЫ»
(АО НПП «РЕЛЭКС»)
Бакинских ул., д. 2Ь, г. Воронеж, 394006
Тел./факс: (473) 2-711-711, 2-778-335
E-mail: market@relex.ru, http://www.relex.ru
ОКПО 45972864, ОГРН 1033600021400,
ИНН/КПП 3664031210/366401001

УТВЕРЖДАЮ
Генеральный директор
АО НПП «РЕЛЭКС»
И. А. Бойченко
«5» 06 2023 г.

АКТ

о внедрении результатов диссертационного исследования
Азиза Аммара Имада Азиза

Комиссия в составе: председателя – технического директора Ефремова Максима Сергеевича и членов комиссии: заместителя генерального директора по развитию Мягковой Ирины Александровны, руководителя проектов Сахарова Владимира Алексеевича, менеджера по персоналу Гурова Сергея Андреевича, настоящим актом подтверждает, что компоненты программного и алгоритмического обеспечения безопасного управления в масштабируемых СУБД, представленные в диссертационном исследовании Азиза Аммара Имада Азиза, использовались в АО НПП «РЕЛЭКС» при разработке отдельных элементов отечественной реляционной системы управления базами данных СУБД ЛИНТЕР.

Председатель комиссии:
Технический директор М.С. Ефремов

Члены комиссии:
Заместитель генерального директора по развитию И.А. Мягкова

Руководитель проектов В.А. Сахаров

Менеджер по персоналу С.А. Гуров

technolo

AL_RAWIYAH FOR IMFORMATION TECHNOLOGY

Al- Rawiyah for information

Baghdad, Almashtal, Iraq

Phone: +964 780 970 7001

Email: alroyla.iq!@gmail.com

Date:1.6.2023

شركة الرؤية
لتقنيات المعلومات والاتصالات

To Whom It May Concern

Act on the implementation of the result of the thesis research by Azeez Ammar Emad,

The team of the company " Al- Rawiyah for information technology" has issued to prove the result of that the result of **Azeez Ammar Emad's dissertation research** about the development of mathematical and software for the process of secure replication management in scalable DBMS, are applied in our company's practical work in design the DBMS, by using the practical implementation of the proposed methodology the performance of private cloud computing environments that accept NoSQL DBMS as a storage system. The models are presented to jointly evaluate throughput and availability, which are important indicators of the quality of our service, and we note that the algorithms used provide an increase in performance and quality of our service with the simultaneous determination of process priorities.

Alhaidary Ammar Jalaaldeen
Chairman

Alhassan Ammar Abdoulameer
Member

Alrubaee Laith hassan
Member

МИНОБРНАУКИ РОССИИ
федеральное государственное
бюджетное образовательное
учреждение
высшего образования
«Воронежский государственный
университет»
(ФГБОУ ВО «ВГУ»)

Университетская пл., 1, Воронеж, 394018.
Тел. (473) 220-75-21, Факс (473) 220-87-55.
E-mail: office@main.vsu.ru
http://www.vsu.ru
ОКПО 02068120, ОГРН 1023601560510
ИНН/КПП 3666029505/366601001

06.06.2023 №0600-254
На № _____ от ____.____.20____

УТВЕРЖДАЮ

Первый проректор –
проректор по учебной работе
ФГБОУ ВО «ВГУ»,
доктор фармацевтических наук,
профессор
Чупандина Е.Е.
«7» июня 2023 г.

АКТ
о внедрении результатов диссертационного исследования Азиза Аммара Имада Азиза

Комиссия в составе: председателя комиссии – декана факультета прикладной математики, информатики и механики, кандидата физико-математических наук, доцента Медведева С.П. и членов комиссии: кандидата физико-математических наук, доцента, доцента кафедры математического обеспечения ЭВМ, заместителя декана по учебной работе Болотовой С.Ю.; кандидата физико-математических наук, доцента, доцента кафедры математического обеспечения ЭВМ Каплиевой Н.А. составила настоящий Акт о том, что результаты диссертационного исследования Азиза Аммара Имада Азиза по разработке математического и программного обеспечения процесса безопасного управления репликациями в масштабируемых СУБД, внедрены в учебный процесс факультета прикладной математики, информатики и механики в программах следующих дисциплин:

- Базы данных (2021-2023 уч. годы, направление 01.03.02 «Прикладная математика и информатика»);
- Информационная безопасность и защита информации (2021-2023 уч. годы, направление 01.03.02 «Прикладная математика и информатика»);

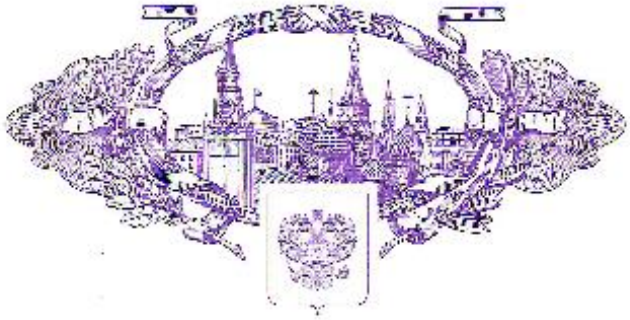
- Информатизация предприятия (2021-2023 уч. годы, направление 01.03.02 «Прикладная математика и информатика»);

Председатель комиссии:
кандидат физико-математических наук, доцент С.П. Медведев

Члены комиссии:
кандидат физико-математических наук, доцент С.Ю. Болотова

кандидат физико-математических наук, доцент Н.А. Каплиева

РОССИЙСКАЯ ФЕДЕРАЦИЯ



СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020664335

«Согласованное распределение ресурсов в интегрированных структурах»

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Воронежский государственный университет» (ФГБОУ ВО «ВГУ») (RU)***

Авторы: ***Бондаренко Юлия Валентиновна (RU), Азиз Аммар Имад (RU), Бондаренко Олег Владимирович (RU)***

Заявка № **2020663633**

Дата поступления **03 ноября 2020 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***11 ноября 2020 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев