САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кравцов Тимур Тихонович

Выпускная квалификационная работа
Исследование самонастраивающихся распределенных систем управления базами данных

Уровень образования: бакалавриат

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5005.2020 «Программирование и информационные технологии»

Научный руководитель:

Корхов Владимир Владиславович, доцент, к.ф.-м.н.

Санкт-Петербург

2024

Содержание

Введение

На сегодняшний день распределенные системы управления базами данных являются стандартом в больших информационных проектах. Они позволяют хранить и обрабатывать огромные объемы данных с повышенной надежностью и эффективностью.

Современные распределенные системы управления базами данных имеют множество конфигурационных настроек. Установка соответствующих значений для этих настроек имеет решающее значение, для обеспечения высокой пропускной способности и низкой задержки распределенной системы управления базами данных (РСУБД). Настройка параметров конфигурации, которые оптимизируют производительность баз данных является NP-трудной задачей. Ручная настройка иногда не справляется с различными нагрузками и аппаратными средами, особенно когда рабочая нагрузка сильно изменяется на протяжении дня. Поэтому автоматическая настройка конфигурации вызывает большой интерес как в научных кругах, так и в промышленности.

Постановка задачи

Цель данной работы состоит в исследовании самонастраивающихся распределенных систем баз данных. Реализация исследования предполагает решение следующих задач:

- Исследование проблемы оптимальной настройки распределенной системы управления базами данных
- Изучение архитектурных решений самонастраивающихся распределенных систем управления базами данных
- Изучение существующих методов самонастройки систем баз данных
- Сравнение методов самонастройки систем баз данных
- Реализация собственной самонастраивающейся распределенной системы базы данных
- Анализ эффективности реализованной системы

Обзор литературы

Работа [4] является одной из главных работ в сфере инструментов для самонастройки аналитической системы баз данных. Инструмент Starfish сочетает в себе множество методов настройки и многоуровневую архитектуру, позволяющую понять основные подходы к решению задачи о настройке. Авторы доходчиво рассказывают о внутреннем устройстве системы сопровождая это иллюстрациями, что помогает восприятию материала.

Статья [5] помогает понять основные компоненты настройки неаналитических систем. Это достигается за счет выделенных компонентов архитектуры в сопровождении с наглядными схемами рабочего процесса системы.

Статья [7] дает обобщающую информацию о методах настройки системы баз данных и выделяет их основные виды.

Работа [8] помогает лучше понять проблемы настройки и существующие решения, ведь авторы подробно разбирают каждый аспект вышеупомянутых тем.

Инструмент [9] является одним из последних инструментов, основанном на эвристических правилах.

Работа [10] содержит информацию о современном инструменте, основанном на Байесовской оптимизации. Его особенность в способности нахождении оптимальных безопасных настроек для системы.

В статье [11] проводится многостраничное описание текущего состояния области инструментов самонастройки аналитических систем больших данных. Ведется обзор множества инструментов, их сильных и слабых сторон, решаемых ими проблем.

Глава 1. Проблема оптимальной настройки распределенной системы баз данных

1.1 Основные свойства распределенных систем баз данных

Под распределенной системой управления базами данных будем понимать программное обеспечение, которое управляет распределенными базами данных и предоставляет механизм доступа к ним. Распределенные базы данных, как правило обладают свойствами, описанными Кристофером Дейтом [1]. Некоторые из них:

- **Локальная автономия.** Это свойство означает, что управление данными на каждом из узлов распределенной системы выполняется локально
- **Непрерывные операции.** Это свойство означает, что данные должны быть всегда доступны, а операции над ними должны выполнятся непрерывно.
- **Прозрачность расположения.** Это свойство означает, что пользователь, обращающийся к распределенной базе данных, ничего не должен знать о реальном, физическом размещении данных в узлах системы.
- **Обработка распределенных запросов.** Это свойство означает возможность выполнения операций выборки над распределенной базой данных.
- **Независимость от оборудования.** Это свойство означает, что в качестве узлов распределенной системы могут выступать компьютеры любых моделей и производителей.

Также на распределенную систему баз данных распространяется CAP теорема [2].

1.2 Распределенные системы управления базами данных и их показатели эффективности

При подборе распределенной системы управления базами данных в существующую информационную систему, специалистам необходимо ответить на следующие вопросы:

- При возникновении перебоев связи в распределенной системе баз данных, может ли пользователь изменять данные или нет?
- Нуждается ли информационная система в механизме транзакций?
- Нуждается ли информационная система в репликации данных, для повышения устойчивости или доступности?
- Каковы требования к масштабируемости информационной системы и технологии масштабируемости системы баз данных?
- Каковы требуемые объемы данных, нуждающиеся в обработке?
- Какая распределенная система баз данных работает с заданной моделью данных?
- Какие возможности восстановления и устойчивости к отказам нужны данной информационной системе?

Ответив на эти вопросы, специалисты выбирают подходящую РСУБД. Таким образом, ответы на эти вопросы могут служить показателями эффективности РСУБД, при условии функционирования внутри заданной информационной системы. Так No-SQL система управления базами данных Apache Cassandra плохо подходит для работы внутри банковской системы, но может быть лучшим решением внутри высоконагруженного интернет-магазина.

После выбора подходящей РСУБД, появляется не менее важная задача по максимизации других доступных показателей эффективности таких как: пропускная способность системы и средняя задержка при ответе на запрос. Повлиять на эти показатели можно по-разному: улучшив физические характеристики самой машины, улучшив физические характеристики каналов

связи внутри сети, оптимизировав работу операционной системы хоста или оптимизировав настройки самой распределенной системы управления базами данных.

1.3 Сложности оптимальной настройки распределенной системы баз данных

Для улучшения вышеописанных показателей эффективности, РСУБД имеет сотни конфигурационных настроек, отвечающих за множество аспектов работы системы. Перечислим некоторые из них:

- Выделение памяти системным процессам и процессам ответственными за работу с клиентами
- Изменение количества кэшируемых данных
- Изменение стоимости операций манипулирования данными, при вычислении оптимального плана выполнения запроса
- Изменение настроек внутренних процессов системы

Множество настроек конфигурации связаны между собой так, что поменяв один параметр, нужно поменять другой, иначе может возникнуть ситуация, где доступные ресурсы системы перерасходованы или недорасходованы. Полный расход ресурсов не гарантирует максимальную эффективность системы. Более того, выбор оптимальных настроек системы зависит от физических характеристик машины, операционной системы хоста, различных характеристик полезной нагрузки, объемов полезной нагрузки, необходимых требований к самому узлу системы. С изменением характеристики полезной нагрузки или ее объемов, оптимальные настройки могут перестать давать высокую эффективность системы. Дополнительно внутри распределенной системы каждый узел требует отдельной настройки.

Из-за вышеописанных причин, задача оптимальной настройки распределенной системы управления базами данных является сложновыполнимой, а при нестандартных или часто изменяющихся полезных

нагрузках даже опытный администратор базы данных может не найти нужных настроек.

Для решения задачи оптимальной настройки РСУБД были разработаны различные методы и программы, которые помогают повысить производительность системы, сэкономить вычислительные и человеческие ресурсы. Как правило, программа настраивает систему в автономном режиме, без необходимости участия администратора, таким образом, полученная в результате система, является самонастраивающейся. Разработки таких методов и систем имеют интерес научного и промышленного характера.

Глава 2. Архитектура самонастраивающихся распределенных систем управления базами данных

2.1 SQL-on-Hadoop системы

SQL on Hadoop — это инструмент, реализующий интерфейс SQL на платформе Hadoop. Он позволяет использовать запросы структурированных данных в стиле SQL для взаимодействия с платформой Hadoop [3] и ее распределенной файловой системой Hadoop Distributed File System (HDFS). Это упрощает внедрение платформы Hadoop в современных информационных системах.

Для рассмотрения архитектуры был выбран один из самых популярных инструментов самонастройки вышеописанной системы – Starfish [4]. На рисунке 1 изображена архитектура системы.

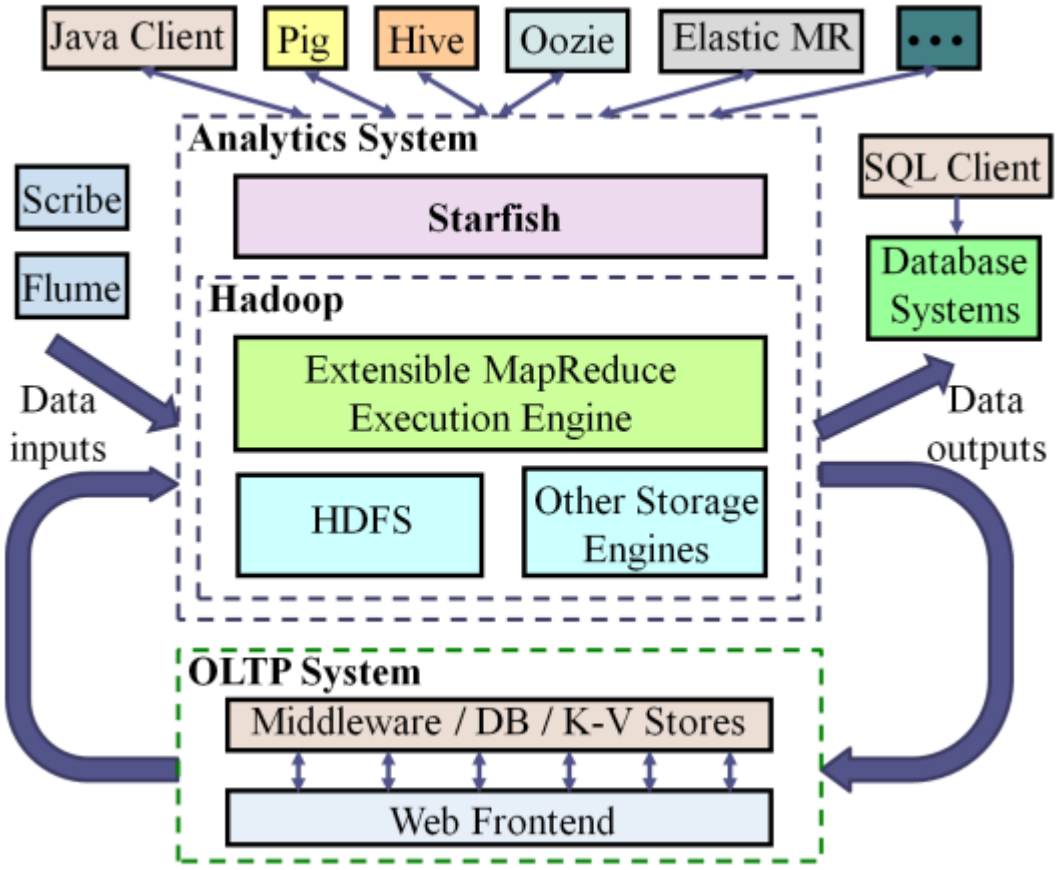

Рисунок 1 Архитектура самонастраивающейся системы Starfish [4 с. 2]

Ядро системы составляет Hadoop, который содержит главные компоненты: движок MapReduce, распределенную файловую систему HDFS и другие движки работы с данными. Starfish работает вместе с Hadoop и образует с ним аналитическую систему. Flume и Scribe собирают, агрегируют и перемещают большие объемы данных из источников в системы хранения данных. Так Flume, Scribe и другие базы данных, обслуживающие веб интерфейсы, служат источниками данных. Запросы для системы могут поступать от Java клиентов, SQL движков Hive или Pig, системами планирования и управления MapReduce заданиями. Полученные аналитические данные загружаются в отдельные базы данных.

Компоненты в Starfish можно разделить на следующие категории:

- Настройка на уровне полезной нагрузки. Эта категория включает в себя оптимизатор нагрузки, эластизатор, «Что если» движок. Эластизатор определяет производительность рабочей нагрузки в заданной конфигурации кластера. И предлагает конфигураций кластера для обработки заданной рабочей нагрузки в соответствии с указанными пользователем целями. Движок "Что если" использует моделирования и симуляции для ответа на различные вопросы выполнения. Движок используется почти всеми оптимизаторами внутри системы.
- Настройка на уровне рабочего процесса. Эта категория включает в себя планировщик, «Что если» движок.
- Настройка на уровне отдельной работы. Эта категория включает в себя «Что если» движок, JIT оптимизатор, профайлер, отборщик проб.
- Менеджеры данных. Эта категория включает в себя менеджера метаданных, менеджера промежуточных данных, менеджера хранения и расположение данных.

Более подробный разбор функциональности этих модулей будет в следующей главе, а пока стоит отметить компоненты, взаимодействующие с системой Hadoop. Оптимизатор нагрузки работает поверх Hadoop и получает входящие запросы и после оптимизации передает их планировщику. Планировщик взаимодействует с менеджерами данных. Также он взаимодействует с планировщиком заданий Hadoop, но работает вне его. Запланированные работы оптимизируется JIT оптимизатором. JIT оптимизатор, находя оптимальные настройки выполнения работы, настраивает соответствующие модули Hadoop. Отборщик проб собирает статистику о входных, промежуточных и выходных ключ-значение пространствах MapReduce работ. Профилировщик использует динамические инструменты для сбора данных о выполнении MapReduce работ. Менеджеры данных собирают необходимую для дальнейшей оптимизации информацию.

В такой модифицированной системе полученный запрос проходит несколько стадий оптимизации в результате чего, он выполняется быстрее и потребляет меньше ресурсов.

Был рассмотрен пример архитектуры самонастраивающейся системы SQL-on-Hadoop и были описаны основные взаимодействия между компонентами системы настройки и Hadoop.

2.2 Реляционные и No-SQL распределенные системы.

Под такими системами будем понимать РСУБД, которые не работают поверх распределенных файловых систем. Приведем несколько примеров: Apache Cassandra, CocoroachDB, кластер MongoDB, кластер PostgreSQL.

В наше время такие системы пользуются большой популярностью, так как являются практичным способом решения множества повседневных задач, которые касаются хранения и манипулирования данными в корпоративных системах. По мере развития информационной системы эти базы данных способны масштабироваться в соответствии с требованиями.

В качестве примера архитектуры был выбран инструмент iBTune [5]. Этот инструмент самонастройки создан для взаимодействия с большими распределенными облачными базами данных. На рисунке 2 представлена архитектура и рабочий процесс iBTune.

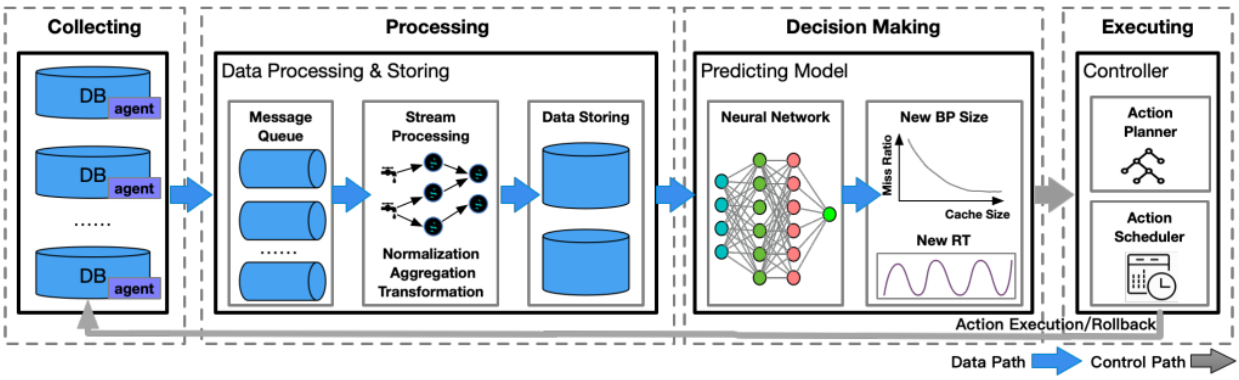

Рисунок 2. Архитектура и рабочий процесс iBTune [5 с. 3].

Можно выделить четыре основных компонента архитектуры: сбор данных, обработка данных, принятие решений, исполнение. В категории сбора данных находятся агенты, ответственные за сбор необходимых метрик и статистики работы отдельных баз данных.

Обработка данных включает в себя систему очередей, систему потоковой обработки и хранилище данных. Система очередей получает данные от агентов и служит для передачи данных между системы потоковой обработки. Система потоковой обработки считывает данные из очереди сообщений и выполняет определенные операции по манипулированию данными. Хранилище данных служит для хранения обработанной информации.

Принятие решений. Этот компонент включает в себя нейронную сеть, которая по данным из хранилища генерирует новые настройки для целевых баз данных.

Исполнение. Этот компонент состоит из планера действий и планировщика действий. Для обработки большого количества баз данных, планер действий стремится создать глобально эффективный и не конфликтующий план выполнения изменений. Это включает в себя настройки

приоритетов между различными категориями действий, объединение действий для одного экземпляра, обнаружение и разрешение конфликтов. Планировщик действий устанавливает определенное время выполнения для всех действий по изменению состояния системы.

IBTune взаимодействует с настраиваемой системой через агентов для сбора информации и контроллера, который производит непосредственное изменение параметров системы. Рабочий процесс iBTune представляет собой замкнутый цикл, поскольку сначала данные собираются из ядра СУБД, обрабатываются и используются для обучения, а затем результаты модели снова применяются к СУБД.

2.3 Выводы.

Рассмотрев вышеописанные примеры, можно прийти к выводам об общей архитектуре самонастраивающихся распределенных систем управления базами данных. Рассмотрим основные категории компонентов и их взаимодействие на Рисунке 3.

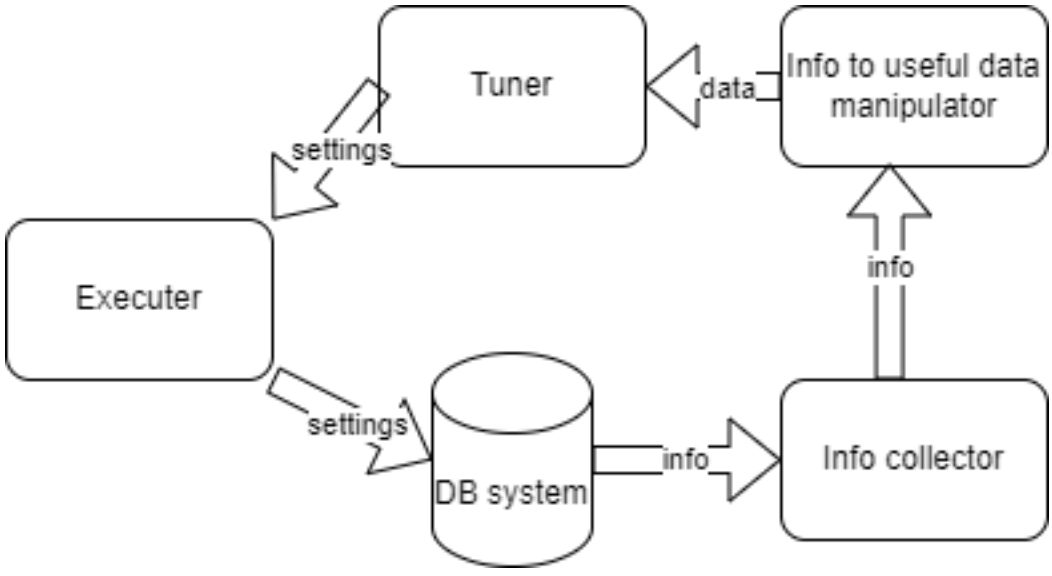

Рисунок 3. Общая архитектура самонастраивющихся РСУБД.

Info collector ответственен за сбор всей необходимой для метода настройки информации. Сложность и количество компонентов внутри этой

категории увеличивается с ростом сложности настраиваемой системы и с увеличением объема данных для метода настройки.

Info to useful data manipulator ответственен за манипуляцию больших объёмов данных о системе, с целью улучшения качества данных и извлечения важных характеристик системы.

Tuner ответственен за применение метода настройки к полученным ранее данным и генерации оптимальных настроек системы. Количество компонентов и их сложность зависит от самого метода настройки системы.

Executor ответственен за изменение конфигурации системы в соответствии с новыми настройками. Большинство систем самонастройки следит за тем, чтобы новая конфигурация оказалась безопасной, поэтому перед применением производится тестирование настроек. Без тестирования в случае ошибки, может использоваться механизм возврата в предыдущее состояние. Также стоит отметить, что в небольших системах практикуется развёртывание еще одной копии системы для тестирования эффективности новых настроек, но такой подход часто является ресурсозатратным для больших систем баз данных.

Вышеописанный набор компонент не является всеобъемлющим, а лишь перечисление необходимого минимума для проектирования и создания любой самонастраивающейся распределенной системы управления базами данных.

Глава 3. Инструменты и методы самонастройки распределенных систем баз данных

3.1 Разбор некоторых реализаций методов и алгоритмов самонастройки

Методы самонастройки являются самой важной частью систем по настройке, ведь качество их работы играет ключевую роль в способности инструмента достигать высокой эффективности. Изучение алгоритмов самонастройки может помочь выделить их основные типы и показать способы имплементации, позволяющие алгоритму решать поставленную задачу. Далее, были разобраны некоторые инструменты, такие как: Starfish [4], iBTune [5].

3.1.1 Алгоритм настройки в Starfish.

Для удобства компоненты Starfish изображены на рисунке 4.

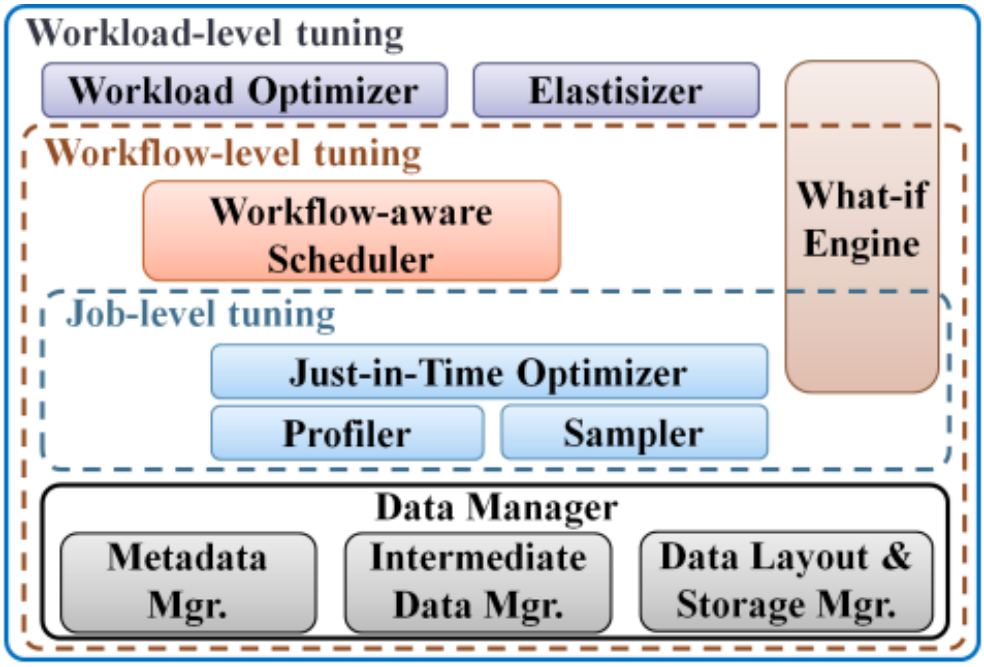

Рисунок 4. Компоненты Starfish [4 с. 3]

Подбор конфигурации настроек производит JIT оптимизатор. Он использует «Что если» движок. Когда Hadoop запускает задание MapReduce, профайлер динамически использует специальные классы Java из пакета

BTrace, для создания профиля задания. Профиль содержит три представления, отражающие различные аспекты выполнения:

- Представление о времени выполнения. Это представление показывает сколько времени было проведено в разных фазах MapReduce задания.
- Представление о потоке данных. Это представление показывает сколько данных было обработано в разных фазах выполнения задания.
- Представление ресурсов. Это представление показывает количество потраченных ресурсов в каждой фазе выполнения.

Профиль помогает диагностировать узкие места при выполнении задания, связанные с используемыми настройками параметров. JIT оптимизатор, взаимодействуя с «Что если» движком, осуществляет оптимизационный поиск по пространству параметров настроек на основе затрат на изменение и прибыли в производительности. Оптимизатор просит движок смоделировать выполнение работы, используя похожий профиль, другие входные данные и настраиваемые параметры. Точный алгоритм оптимизационного поиска оптимизатора не описан в работе. Если подходящего профиля нет, то с помощью отборщика проб генерируется виртуальный профиль работы.

Настройка на уровне рабочего процесса осуществляется планировщиком Starfish. Он обеспечивает координацию политик оптимизации работ с политиками размещения данных, используемыми распределенной файловой системой Hadoop. Вместо того, чтобы принимать локально оптимальные решения для отдельных работ MapReduce, планировщик принимает решения с учетом взаимосвязей между выходными и входными данными выполняемых работ. Планировщик рабочего процесса Starfish работает совместно с движком "Что если" и оптимизатором JIT для выбора оптимального расписание выполнения, а также оптимального расположения

данных. В пространство оптимизации входят следующие параметры: политика расположения блоков данных, количество реплик записи, размер блока файла данных, сжатие данных. Планировщик выполняет поиск оптимального формата выходных данных для каждой работы в данном рабочем процессе. Движок «Что если», получая набор работ и настройки кластера, моделирует выполнения работ и показывает время выполнения и расположение данных. Планировщик использует полученные ответы для определения затрат и выгод при различных вариантах выбора.

Настройкой на уровне полезной нагрузки занимается оптимизатор полезной нагрузки. Существуют три важные категории оптимизации запроса: переиспользование промежуточных данных, единичное вычисление одинакового набора данных для нескольких рабочих процессов и сохранение данных в более удобных местах. Оптимизатор представляет запрос в виде ориентированного графа и применяет оптимизации в виде преобразований графа. Вычислением эффективности нового рабочего процесса занимается «Что если» движок, получая сам граф запроса и настройки кластера.

Таким образом, все оптимизаторы Starfish используют моделирующие способности «Что если» движка для поиска оптимальных настроек, ориентируясь на отношение временных и ресурсных затрат к выигрышу эффективности. Можно предположить, что поиск осуществляется благодаря дереву поиска или случайного рекурсивного поиска. Так, улучшив один параметр, оптимизатор пытается оптимизировать связанный с ним другой параметр, а не перебирает случайный набор параметров вне зависимости от предыдущих результатов. Это напоминает обучение с подкреплением, где модель, взаимодействуя со средой, получает ответ и меняет оптимизируемые параметры.

3.1.2 Алгоритм настройки в iBTune.

Система iBTune занимается настройкой буферного пула индивидуальных узлов системы, основываясь на показателях кэш промаха

сервера. Для начала, каждому узлу сопоставляется описательный вектор, с помощью которого определяются кластеры похожих узлов. Описательный вектор составляется с использованием низкоуровневых характеристик узла и показателей производительности: запросы в секунду, время ответа, использование ЦПУ, время записи и чтения. Так как показатели производительности могут меняться, агенты узлов производят замеры на протяжении множества дней, пока показатели не стабилизируются.

Далее, модуль обработки, составив вектор описания, сохраняет его в распределенное хранилище данных для дальнейшего использования. Используя законы связи размера буферного пула и кэш промаха, можно вычислить более лучший теоретический буферный пул, но полученное значение может ухудшить производительность узла [5, 6]. Формула 1 демонстрирует это отношение.

$$\frac{\log(mr_{target}) - \log(mr_{cur})}{log(bp_{target}) - \log(bp_{cur})} \approx a_i$$

Формула 1. Закон связи кэш промаха и размера буферного пула

Здесь $mr_{target/cur}$ – целевое / текущее отношение кэш промаха, $bp_{target/cur}$ – целевой / текущий буферный пул, a_i – степенной закон распределения запросов.

Во избежания ухудшения производительности после изменений, в компоненте принятия решений глубокая нейронная сеть пытается предсказать новое время отклика. Она получает описательный вектор самого узла и описательный вектор его похожего соседа с текущим показателем времени отклика. Так происходит для всех соседей узла. Итоговое значение отклика усредняется и сравнивается с безопасным и допустимым.

Полученное изменение планируется к применению в компоненте планирования. В случае утраты производительности узла, планер осуществляет откат.

Инструмент iBTune реализовал алгоритм настройки, основанной на математическом вычислении оптимального значения буфера обмена, а для проверки этого значения используется обученная глубокая нейронная сеть. Из сильных сторон можно отметить высокую отказоустойчивость полученной системы из-за нескольких механизмов обеспечения безопасности конфигураций, и робастность к изменениям самих характеристик системы за счет обучения на агрегированных за многие дни данных. Слабой стороной является наличие больших вычислительных ресурсов для тренировки вышеупомянутой нейронной сети.

3.2 Основные методы самонастраивающихся систем баз данных.

Для нахождения или принятия решения по применению новых настроек для системы баз данных, множество систем самонастройки используют эвристические методы настройки, методы настройки на основе байесовской оптимизации, методы настройки на основе глубокого обучения, методы настройки на основе обучения с подкреплением [7]. Аналитические системы баз данных, как правило, используют методы на основе моделирования или симулирования.

3.2.1 Эвристические методы

Эвристические методы можно разделить на методы, основанные на правилах, и методы, основанные на поиске.

Так, методы, основанные на правилах, производят настройку на основании правил из опыта администраторов баз данных или мануалу настройки. Они используют заранее заданные правила настройки или вопросы вида "Что, если?" для настройки производительности. Похожий метод реализован в разобранном выше инструменте Starfish. Методы, основанные на правилах, могут ускорить процесс принятия решений по сравнению с администраторами баз данных.

«Преимущество правил, что они просты и быстры и могут повысить производительность, заменив администраторов баз данных эффективными правилами настройки. Однако они могут не найти оптимальную конфигурацию, поскольку на настройку регулятора влияет множество факторов, и с помощью простых правил трудно добиться оптимальной настройки». [8 с. 10-11]

Методы, основанные на поиске, разделяют пространство конфигурации на несколько подпространств. Затем, запускают заданную рабочую нагрузку в этом подпространстве конфигурации и выбирают подпространство с лучшими показателями эффективности. Затем, при необходимости выполняют поиск по соседям этого подпространства, чтобы найти более подходящие настройки. Один из последних разработанных инструментов: BestConfig[9].

«Хотя метод настройки на основе поиска не может гарантировать глобально оптимальную настройку регулятора, он позволяет найти почти оптимальную конфигурацию, которая обычно близка к наилучшим возможным настройкам регулятора базы данных. По сравнению с методом настройки, основанным на правилах, методы, основанные на поиске, не опираются на исторический опыт и могут адаптироваться к различным системам баз данных. Тем не менее, они имеют ряд существенных недостатков. Во-первых, процесс выборки и поиска занимает много времени. Во-вторых, в процессе выборки также могут отсутствовать лучшие настройки конфигурации, поскольку некоторые области поиска могут быть проигнорированы». [8 с. 11]

3.2.2 Методы на основе Байесовской оптимизации

Байесовская оптимизация (БО) — это метод, используемый для оптимизации сложных и дорогостоящих целевых функций. В отличие от традиционного поиска по сетке или случайного поиска, байесовская оптимизация использует вероятностную модель для оценки поведения целевой функции и управления процессом поиска, который обеспечивает

баланс между исследованием и эксплуатацией для эффективного определения оптимума. Как правило, БО инициализирует модель с помощью нескольких выбранных точек данных. На каждой итерации БО использует функцию сбора данных, чтобы решить, какие новые точки выбрать, в соответствии с политикой разведки. Чем больше итераций проходит, тем более точно аппроксимируется целевая функция оптимизации. Среди работ по самонастройке, основанных на БО, гауссовский процесс является наиболее популярной моделью, так как им легко аппроксимировать различные распределения. Современный инструмент, использующий данный метод: OnlineTune[10].

«Методы настройки, основанные на БО, как правило, превосходят эвристические методы. Однако методы, основанные на БО, особенно те, которые используют гауссовский процесс, не могут эффективно масштабироваться до большого конфигурационного пространства и оказываются неоптимальными. Кроме того, для суррогатных моделей в методах настройки, основанных на БО, гауссовский процесс в основном подходит для оптимизации непрерывных настроек; в то время как SMAC может поддерживать как непрерывные, так и категориальные настройки и работает лучше, чем гауссовский процесс для относительно большого конфигурационного пространства. Однако гауссовский процесс может эффективно исследовать нетронутое конфигурационное пространство, для которого древовидная модель в SMAC не может обещать высокого качества прогнозирования». [8 с.12]

3.2.3 Методы на основе глубокого обучения

В методах на основе глубокого обучения в качестве модели настройки используется глубокая нейронная сеть, которая по входным данным пытается предсказать оптимальные настройки. Конечно, такой метод обладает известными плюсами и минусами.

Нейронные сети могут более точно улавливать особенности и сложность системы, в отличие от других методов, но для их обучения нужно собрать больше всего данных и затратить больше всего вычислительных ресурсов. Порою система еще не располагает необходимыми данными и ресурсами. Также необходимо решить задачу выделения главных характеристик системы, для ускорения обучения и способ передачи знаний, на случай изменений важных характеристик. Из рассмотренных инструментов глубоким обучением пользуется iBTune[5].

3.2.4 Методы на основе обучения с подкреплением

В обучении с подкреплением модель или агент в качестве обучения использует метод проб и ошибок при взаимодействии с внешней средой, которая дает вознаграждения. Суть метода заключается в изучении стратегии настройки на основе взаимодействий между средой и агентом. Так, информационная система используется как внешняя среда, показатели эффективности выполнения воспринимаются как состояния, модель настройки используется как агент, настройка параметров воспринимается как действия, изменение производительности после настройки используется как награда, а модель оценки настройки определяет политику действий.

Из устройства системы можно понять, что данный метод хорошо подходит в случаях, когда нужно произвести настройку сложной системы, но данных для обучения нейронных сетей не хватает. Метод может исследовать большие пространства настройки, но за это нужно заплатить многократными взаимодействиями с системой, долгим процессом вычисления следующих оптимальных настроек и возможностью произвести небезопасную настройку. Поэтому как правило нужно реализовывать дополнительный механизм обеспечения безопасности состояния системы. Методы, основанные на БО, быстрее приходят к оптимальным настройкам, но из-за кубической сложности, чем больше итераций проходит, тем лучше и относительно быстрее будут работать методы обучения с подкреплением.

3.2.5 Методы на основе моделирования или симулирования

Моделирование затрат — это метод, который использует набор математических формул для прогнозирования производительности. Оптимизационный алгоритм использует модель системы для нахождения оптимального решения. Параметры модели обычно рассчитываются на основе информации журнала или выборочных заданий. В пример инструмента, использующего данный метод, можно привести описанный ранее Starfish [4].

Такие модели очень эффективны в вычислительном отношении для поиска правильных настроек параметров, а время настройки составляет от миллисекунд до нескольких секунд, в зависимости от используемого алгоритма оптимизации. С другой стороны, модели часто основаны на упрощенных допущениях, таких как предположение о пропорциональности размера входных и выходных данных работы, или на том, что все узлы кластера однородны. Наконец, подходы, основанные на затратах, обычно моделируют определенную версию фреймворков, и их трудно адаптировать после внесения изменений в механизм выполнения. [11]

Иногда можно использовать подход, основанный на симулировании. Хоть такой метод является более затратным по времени и ресурсам, он более детально отражает тонкости системы при изменении характеристик компонентов системы и рабочей нагрузки.

3.3 Обзор некоторых инструментов самонастройки систем баз данных

На рисунках 5, 6, 7, представлен обзор инструментов самонастройки систем баз данных по их методам и подходам.

Category	Work	Tuning Objective	Knob Selection	Feature Selection	Tuning Method	Transfer Techniques
Heuristic (Rule-based)	Probability Theory [69]	Performance	Tuning experience	N/A	Probability theory and decision theory	N/A
	PGTune [1]	Performance	Tuning experience	N/A	Empirical rules	N/A
Heuristic (Search-based)	OpenTuner [5]	Performance	Tuning experience	N/A	E.g., multi-armed bandit, particle swarm optimization	N/A
	Bestconfig [98]	Performance	Tuning experience	N/A	Configuration sampling (DDS) + search algorithm (RBS)	N/A
BO-based	iTuned [73]	Performance	Sensitivity analysis	N/A	Gaussian process	N/A
	OtterTune [79]	Performance	Lasso	Factor Analysis K-Means	Gaussian process	Workload mapping
	RelM [45]	Performance	Tuning experience	N/A	Gaussian process	N/A
	ResTune [92]	Performance Resource utilization	Tuning experience	Occurrence frequency	Gaussian process	Model ensemble
	CGPTuner [10]	Performance	Tuning experience	N/A	Contextual Gaussian process	Workload mapping
	OnlineTune [93]	Performance	Tuning experience	Query/Data characterization	Contextual Gaussian process	Model ensemble
	LlamaTune [40]	Performance	Tuning experience	N/A	Gaussian process; SMAC	N/A
DL-based	iBTune [73]	Performance	Tuning experience	N/A	Convolutional Neural Network	N/A
	DNN [80]	Performance	Lasso	Factor Analysis K-Means	Deep Neural Network	N/A
RL-based	CDBTune [89]	Performance	Tuning experience	Read/Write ratio	Deep Deterministic Policy Gradien (DDPG)	N/A
	QTune [52]	Performance	Tuning experience	Query character-ization	Double-State DDPG	Workload embedding
	UDO [82]	Performance Resource utilization	Tuning experience	N/A	A MCTS algorithm with delayed feedback	N/A
	WATuning [26]	Performance	Tuning experience	Read/Write ratio	Attention + DDPG	Workload embedding
	DB-BERT [77]	Performance	NLP (BERT)	N/A	Language model + DDQN	N/A
	HUNTER [9]	Performance	Random forest	PAC	Genetic algorithm + DDPG	N/A

Рисунок 5. Обзор инструментов настройки систем баз данных [8 с. 15].

Как видно из рисунка 5, только малая часть инструментов берут во внимание экономию расходуемых ресурсов. Большинство инструментов предпочитают выбирать настраиваемые параметры из собственного опыта. Большинство методов, основанных на машинном обучении, предпочитает преобразовывать данные системы в признаки, в то время как эвристические методы – нет. На удивление около половины методов, основанных на машинном обучении, не используют никакой техники передачи знаний.

	Approach	Profiling	Prediction	Optimization
Hadoop MapReduce	Starfish [58, 61]	Dynamic instrumentation	Analytical models, black-box models, and simulation	Recursive Random Search
	Predator [121]	Dynamic instrumentation	Analytical models	Grid Hill Climbing
	MRTuner [102]	Information from logs	Producer-Transporter-Consumer analytical model	Grid-based Search
	MR-COF [84]	Dynamic instrumentation	Analytical models/MRPerf [118] simulations	Genetic Algorithm
	ARIA [114]	Information from logs	Analytical models	Lagrange Multipliers
	Zhang et al. [133]	Dynamic instrumentation	Platform- and job-level analytical models	N/A
	HPM [116]	Information from logs	Scaling analytical models and linear regression	Brute-force Search
	IHPM [73]	Information from logs	Scaling models and locally weighted linear regression	Lagrange Multipliers
	CRESP [26]	Information from logs	Analytical models and linear regression	Brute-force Search
	Elastisizer [60]	Dynamic instrumentation	Same as Starfish and M5 regression-tree model	Recursive Random Search
Spark	Wang [120]	Sample job execution	Analytical model	N/A
	Ernest [113]	Sample job execution	Parametric model & NNLS	N/A
	Dione [131]	Sample job execution and Graph Edit Distance	Parametric model & NNLS	N/A
	Singhal [103]	Sample job execution	Analytical models	Grid Search
	DynamiConf [46]	Profile benchmarks	Parametric models	Iterated Local Search
	Chen et al. [28]	Sample job execution	Analytical models	Range Search
Storm	Sax et al. [101]	Performance metrics	Analytical model	Direct Algorithm
	Bedini et al. [20]	Performance metrics	Fine-grained cost model	N/A
Heron	Trevor [17]	Performance metrics	Linear models & Net flow	N/A
	Caladrius [71]	Performance metrics	Timeseries & Cost models	Topological Sorting

Рисунок 6. Обзор современных инструментов настройки аналитических систем [9 с. 8].

Из таблицы настройщиков аналитических систем можно увидеть, что для настройки Hadoop, около половины инструментов используют логи. Другая же половина утилизирует динамический инструментарий. Для Spark большинство инструментов запускает целевой запрос на малом кластере или запускает лишь часть работ запроса. Полученная от запусков информация, помогает предсказать поведение запроса во время его полноценного выполнения. Также стоит отметить наличие у Starfish самого богатого инструментария для вычисления оптимальных настроек.

Approach	Input Features	ML Technique
Kavulya et al. [72]	Job submission time, job name, user name, number of map/reduce tasks, and map input bytes/records	K-Nearest-Neighbors
	Number of map/reduce tasks and map input bytes/records and weights based on job similarity	Locally weighted Linear Regression
Kadirvel et al. [70]	Number of nodes, map input size, number of reduce tasks, I/O sort record percent, and the shuffle parallel copies	Gaussian Processes, Multilayer Perception, Discretization regressor, M5 Pruned Model Tree
Yigitbasi et al. [128]	`mapreduce.task.io.sort.mb`, number of map and reduce slots, number of reduce tasks, and map input size	Support Vector Regression
AROMA [76]	CPU, disk, and network utilization patterns	k-mediod Clustering
	Input data size, resource allocations, and several parameters	Support Vector Machine
PPABS [124]	CPU, memory, and disk utilization statistics	k-means++ Clustering
	CPU, memory, and disk utilization statistics	Pattern Recognition
PStorM [40]	Features from code analysis (e.g., map class name) and execution profile (e.g., map size selectivity)	Gradient Boosted Regression Trees
Chen et al. [25]	A set of several configuration parameter values	Tree-based Regression
RFHOC [21]	A set of several configuration parameter values	Random-Forest Approach
Wang et al. [119]	CPU cores, memory, max map output, compress, shuffle, block size, parallelism, input data size, etc.	Decision tree (C5.0), Recursive Random Search
Hernández et al. [55]	CPU cores, memory, parallelism	Gradient Boosting Regressors
PBDST [67]	Simultaneous multi-threading (SMT) configurations	k-means, logistic regression
d-Simplexed [27]	CPU, memory, data size	Delaunay Triangulation
Zacheilas et al. [130]	A set of several configuration parameter values	Gaussian Processes
Li et al. [79]	Memory sizes and number of cores & threads in various stages	Support Vector Regression
Trotter et al. [108]	Number of worker processes, number of executors	Genetic Algorithm, Bayesian Optimization
Trotter et al. [109]	Workers, spout parallelism, bolt parallelism, acker parallelism	Genetic Algorithm, Support Vector Machines
OrientStream [117]	Variety of data-, plan-, operator-, and cluster-level features	Ensemble/Incremental ML
Vaquero et al. [110]	Parameters/metrics selected based on Factor Analysis	Reinforcement Learning

Рисунок 7. Обзор современных инструментов настройки аналитических систем [9 с. 13].

Из таблицы на рисунке 7 можно заметить, что обозреваемые методы используют самые разные подходы. Начиная от линейной регрессии и заканчивая генетическими алгоритмами. Большинство из обозреваемых методов используют данные о загрузке процессора, оперативной памяти, входных данных запроса и наборе из задач.

3.6 Выводы

В данной главе были исследованы основные методы самонастраивающихся распределенных систем управления базами данных. Для изучения реализаций методов, были разобраны архитектуры некоторых инструментов самонастройки. Было проведено сравнение методов между собой и описаны их преимущества и недостатки. Также, рисунки с обзорами

инструментов помогают сформировать представление о современном состоянии исследований и работ в данной области. Приведенная в главе информация позволяет понять общее внутреннее устройство самонастраивающихся систем, что позволяет спроектировать собственную самонастраивающуюся РСУБД и осуществить ее реализацию.

Глава 4. Реализация самонастраивающейся распределенной системы управления базами данных

4.1 Архитектура системы

В качестве распределенной базы данных был выбран высокодоступный кластер Postgresql с одним главным узлом и двумя последователями. Такой выбор обоснован приемлемым количеством параметров настройки узлов базы данных. Пространство настройки в других системах было бы в разы больше. Высокодоступность подразумевает отсутствие единой точки отказа в системе, кроме точки входа.

Из доступных инструментов для создания подобной системы был выбран шаблон для создания высокодоступных кластеров Postgresql – Patroni [12]. Patroni, используя стриминговую репликацию Postgresql, создает и поддерживает кластер. Он ответственен за восстановление отдельных узлов, поддержку согласованности, и переконфигурирования настроек кластера и узлов. Patroni включает в себя поддержку балансировщика нагрузки HAproxy, и распределенной системы контроля ETCD, Zookeeper, Consul.

В качестве системы контроля был выбран ETCD. Кластер ETCD, работая на основе протокола консенсуса RAFT, ответственен за согласованный выбор лидера из узлов Postgresql. Так, при отказе лидера, ETCD перестает получать сердцебиение и объявляет гонку за лидерство среди живых последователей. Один из узлов производит захват лидерства, а старый лидер получает отказ при попытке возобновить правление.

HAproxy обеспечивает буферизацию и распределение запросов, так если нужный узел не сможет ответить на запрос, он не вернется с ошибкой а будет удерживаться некоторое время HAproxy. Рабочий процесс системы изображен на рисунке 8.

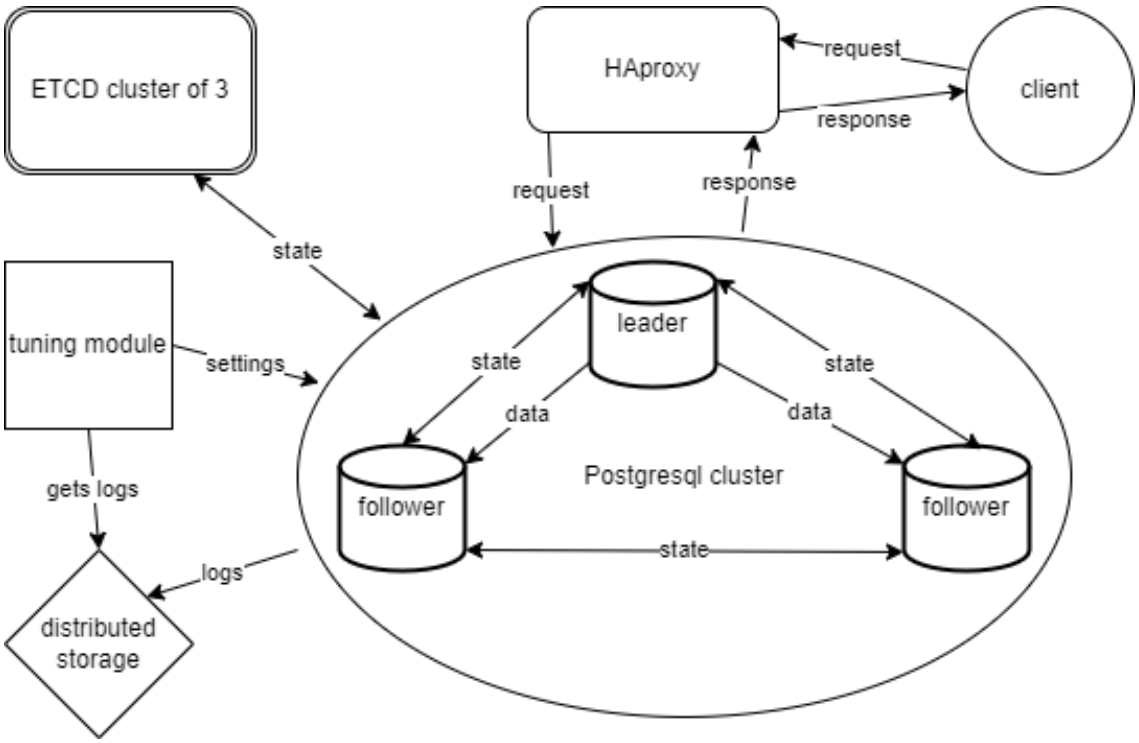

Рисунок 8. Архитектура и рабочий процесс разрабатываемой системы

Для функционирования компоненты настройки, узлы системы баз данных записывают логи запросов в распределенную файловую систему. Периодически компонента настройки считывает логи и производит настройку каждого узла, если это необходимо. Распределенная файловая система может быть заменена на локальную файловую систему при реализации для локального запуска.

Было принято решение сделать основной метод настройки на основе правил, ведь они производят настройку системы быстрее, выдают безопасные настройки и не требуют обучения и ресурсов для поиска оптимальных настроек. С другой стороны, полученные настройки не являются самыми оптимальными. Тем не менее, такой модуль позволит получить улучшения эффективности системы за минимальную плату. Существует достаточно правил для настройки Postgresql, которые помогут в создании алгоритма подбора настроек. Далее в работе будет проведен замер эффективности методов настройки на локальной небольшой системе, и можно будет сравнить полученные результаты разных методов.

Сама компонента настройки состоит из следующих модулей: connecter, data getter, executor, query parser, stat manager, tuner.

- connector – отвечает за соединение с узлами систем баз данных
- data getter – отвечает за получение новейших логов
- executor – отвечает за применение новых настроек к узлам
- query parser – отвечает за извлечение данных из логов.
- stat manager – отвечает за хранение и изменение внутреннего состояния модуля
- tuner – отвечает за хранение и применение нужных правил, основываясь на состоянии системы.

4.1 Реализация системы

Система была реализована с Patroni версии 3.3.0, которая была клонирована из github [13]. Версия запускаемых узлов Postgresql – 16.0. Язык реализации системы настройки – Python 3.12.0. Использованные библиотеки: aiopg для асинхронного подключения к узлам postgresql, async-timeout и psycopg2-binary для работы aiopg.

Модуль tuner состоит из класса Tuner с методами get_knobs и tune. Метод get_knobs возвращает словарь с параметрами настройки. Суть метода tune основывается на нескольких правилах. Если поступающие запросы являются преимущественно чтением, то мы хотим забрать допустимое количество оперативной памяти у служебных процессов, ответственных за очистку системы, ведущих статистику системы, и от размеров журнала записи. Все это сделано для увеличения доступной памяти для кэширования запрошенных данных. Также можно помочь дополнительной памятью процессам сессии, для хранения временных таблиц или вычисления полей. Такое перераспределение памяти непременно дает выигрыш эффективности, если размер обслуживаемой базы данных во много раз не превышает оперативную память системы. Во время преимущественной записи все происходит наоборот. Таким образом метод tune получает на вход текущее соотношение запросов

чтения к запросам записи и на основе этого пропорционально распределяет память системы и передает новые параметры модулю executor.

Модуль executor состоит из класса Executor и метода execute. Executor инициализируется с помощью готового соединения с узлом из класса ConnectorPG. Метод execute подключается к СУБД и исполняет SQL команды для изменения настроек.

Модуль connector состоит из класса ConnectorPG с методами create_pool, get_connection, close_pool. ConnectorPG инициализируется с помощью данных для подключения к заданным узлам Postgresql. Метод create_pool создает пул соединений. Метод get_connection возвращает соединение из пула. Метод close_pool закрывает созданный ранее пул соединений.

Модуль data getter состоит из класса DataGetter с методом get_file_name. DataGetter инициализируется с помощью данных о расположении логов. Метод get_file_name возвращает имена новых логов.

Модуль stat manager состоит из класса Stat_Manager и методов add_update, add_select, asses_dbrw, get_new_knobs. Stat_Manager во время инициализации создает экземпляр класса Tuner и словарь для хранения количества записей и чтения. Метод asses_dbrw проверяет отношения чтение к записи с предыдущим отношением и при изменении более чем на 0.2 вызывает Tuner для получения новых настроек. Метод get_new_knobs возвращает названия настроек.

Модуль query parser состоит из класса QueryParser и метода parse. QueryParser инициализируется существующим экземпляром класса Stat_Manager. Метод parse принимает путь к логу и передает количество найденных команд UPDATE и SELECT в словарь Stat_Manager.

Полный код системы можно найти по ссылке на github [14].

4.3 Анализ эффективности реализованной системы

При анализе эффективности системы, в качестве показателей эффективности было выбрано количество транзакций в секунду и время на

ожидание ответа. Система была протестирована различными инструментами и следующими нагрузками: TPC-B like из инструмента pgbench [15], read oltp из инструмента Sysbench [16], write oltp из инструмента Sysbench. Также для сравнения были проведены замеры системы, настроенной результатом работы метода обучения с подкреплением DDPG из репозитория [17]. Метод был заранее обучен на данных Sysbench write.

TPC-B like нагрузка имитирует активность клиента интернет магазина с обновлениями, записями и чтениями.

Read oltp \ write oltp имитирует клиента отправляющего несколько запросов на чтение \ запись для нескольких таблиц.

При нагрузке система была запущена в Docker [18], где каждый компонент архитектуры имел свой контейнер. Sysbench был запущен из WSL 2.0, так как не поддерживается системой Windows. Размер каждой из нагрузочных баз данных был около 1-го гигабайта. Характеристики системы: Windows10, Intel(R) Core(TM) i3-8130U CPU @ 2.20ГГц 2.20 ГГц, RAM 6 Гб. Нагрузки длились на протяжении 15 минут.

Таблица 1. Измерение транзакций в секунду

	Patroni cluster с заводскими настройками	Patroni cluster с заводскими настройками с модулем настройки	Patroni cluster с настройками метода DDPG обученном на данных OLTP write
OLTP Read	Среднее количество транзакций в секунду: 2204.41	Среднее количество транзакций в секунду: 2264.82	Среднее количество транзакций в секунду: 2231
OLTP Write	Среднее количество транзакций в секунду: 877	Среднее количество транзакций в секунду: 876.45	Среднее количество транзакций в секунду: 893.21
TPC-B	Среднее количество транзакций в секунду: 172	Среднее количество транзакций в секунду: 180.	Среднее количество транзакций в секунду: 185.5

Таблица 2. Измерение времени ответа

	Patroni cluster с заводскими настройками	Patroni cluster с заводскими настройками с модулем настройки	Patroni cluster с настройками метода DDPG обученном на данных OLTP read
OLTP Read	Средняя скорость ответа: 29.02мс	Средняя скорость ответа: 27.51мс.	Средняя скорость ответа: 28.66мс.
OLTP Write	Средняя скорость ответа: 72.87мс	Средняя скорость ответа: 73.00мс.	Средняя скорость ответа: 71.12мс.
TPC-B	Средняя скорость ответа: 368.407 мс	Средняя скорость ответа: 354.86мс.	Средняя скорость ответа: 344.571мс.

Как видно из эксперимента показатели эффективности были улучшены. Хотя и показатели DDPG превосходят показатели метода правил при записи, при чтении правила хорошо оптимизируют систему и имеют преимущество.

Так как физические характеристики локального компьютера скромны, во время нагрузок, что при заводских настройках, что при настройках модуля, система быстро упирается в узкое горлышко количества оперативной памяти или количества процессоров. Это непосредственно уменьшает потенциал улучшения эффективности системы. Поэтому полученные результаты можно считать вполне приемлемыми.

4.4 Выводы

С помощью познаний архитектур и основных методов настройки РСУБД была спроектирована и разработана самонастраивающаяся распределенная система управления базами данных. Были проведены различные замеры эффективности, показавшие относительно небольшой, но стабильный прирост к главным показателям эффективности. Полученные результаты отражают тот факт, что в небольших системах методы основанные на правилах способны производить настройку на ровне с более продвинутыми методами. Стоит

отметить, что система была запущена локально и каждый узел даже при заводских настройках имел доступ ко всем ресурсам системы. Тогда полученный прирост говорит о более грамотном распределении ценной оперативной памяти и вычислительно мощи процессора.

Заключение

Было проведено исследование самонастраивающихся распределенных систем управления базами данных. Исследованы проблемы оптимальной настройки распределенной системы управления базами данных. Изучены существующие методы самонастройки систем баз данных. Изучены архитектурных решений самонастраивающихся распределенных систем баз данных. Произведено сравнение методов самонастройки систем баз данных. Реализована собственная самонастраивающаяся распределенная система баз данных. Произведен анализ эффективности реализованной системы. Все задачи, сформулированные перед началом работы, были выполнены в полном объёме. Поставленные цели в ходе работы были достигнуты.

В дальнейшей работе стоит произвести улучшение алгоритма самонастройки дополнительными методами оптимизации, и реализовать его на более эффективном языке программирования.

Список литературы

[1] An introduction to database systems 8th edition by Date, C. J.

[2] Eric A. Brewer. Towards robust distributed systems. (Invited Talk) Principles of Distributed Computing, Portland, Oregon, July 2000.

[3] Официальный сайт Hadoop. [Электронный ресурс] URL: https://hadoop.apache.org (дата обращения 23.05.2024)

[4] Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. 2011. Starfish: A self-tuning system for big data analytics. In Proceedings of the 5th Biennial Conference on Innovative Data Systems Research (CIDR'11). 261–272

[5] J. Tan, T. Zhang, F. Li, and et al. ibtune: Individualized buffer tuning for large-scale cloud databases. *VLDB*, 2019

[6] C. Berthet. Approximation of LRU caches miss rate: Application to power-law popularities. arXiv:1705.10738, 2017

[7] Z. Xinyi, C. Zhuo L. Yang, W. Hong, L. Feifei, C. Bin. Facilitating Database Tuning with Hyper-Parameter Optimization: A Comprehensive Experimental Evaluation. Proceedings of the VLDB Endowment. 1808–1821. 2022

[8] X. Zhao, X. Zhou and G. Li, "Automatic Database Knob Tuning: A Survey," in IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 12, pp. 12470-12490, 1 Dec. 2023

[9] Y. Zhu and et al. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In *SoCCer*, 2017

[10] X. Zhang, H. Wu, Y. Li, and et al. Towards dynamic and safe configuration tuning for cloud databases. In *SIGMOD*, 2022

[11] H. Herodotos, C. Yuxing, L. Jiaheng. A Survey on Automatic Parameter Tuning for Big Data Processing Systems. ACM Computing Surveys. 1-37. 2020

[12] Документация Patroni [Электронный ресурс]. URL: https://patroni.readthedocs.io/en/latest/README.html (дата обращения 23.05.2024)

[13] Официальный репозиторий Patroni [Электронный ресурс]. URL: https://github.com/zalando/patroni/tree/master (дата обращения 23.05.2024)

[14] Репозиторий с кодом реализации системы [Эдектронный ресурс]. URL: https://github.com/Lixerus/self-tune-postgreSQL-cluster (дата обращения 23.05.2024)

[15] Официальная документация pgbench [Электронный ресурс]. URL: https://www.postgresql.org/docs/current/pgbench.html (дата обращения 23.05.2024)

[16] Открытый репозиторий с кодом Sysbench [Электронный ресурс]. URL: https://github.com/akopytov/sysbench (дата обращения 23.05.2024)

[17] Открытый репозиторий с кодом DDPG [Электронный ресурс]. URL: https://github.com/evolveDB/tuning-survey/tree/main (дата обращения 23.05.2024)

[18] Официальный сайт Docker [Электронный ресурс]. URL: https://www.docker.com (дата обращения 23.05.2024)