Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки: 09.03.04 – Программная инженерия

Профиль: Технологии разработки информационных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА ИНСТРУМЕНТА ДИНАМИЧЕСКОЙ И РЕКОМЕНДАТЕЛЬНОЙ КОНФИГУРАЦИИ БАЗЫ ДАННЫХ POSTGRESQL

Обучающийся 4 курса

группы 11-706 Задыкян А. А.

Научный руководитель

д.н., профессор, профессор

кафедры программной инженерии Елизаров А. М.

Директор ИТИС КФУ

канд. техн. наук Абрамский М.М.

Казань – 2021

СОДЕРЖАНИЕ

ГЛОССАРИЙ

Облачные сервисы AWS (Amazon Web Services) — платформа удаленного доступа к центрам обработки данных (ЦОД), хранилищам баз данных, серверам и веб-приложениям

Представление — структура данных в СУБД, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению

СУБД — это программное обеспечение или служба, используемая для создания и управления базами данных

HDD (Hard Disk Drive) — запоминающее устройство. Является основным накопителем данных в большинстве компьютеров

PostgreSQL — это система управления реляционными базами данных (СУРБД) с открытым исходным кодом и моделью клиент-сервер

RDBMS (Relational Database Management System) - это система для управления базами данных, базирующаяся на реляционной модели

Vacuum — процесс очистки таблиц и индексов в PostgreSQL, предотвращающий неконтролируемый рост занимаемой памяти

WAL (Write-Ahead Log) — журнал предварительной записи, используемый в PostgreSQL для сохранения целостности данных при возникновении нештатных ситуаций

ВВЕДЕНИЕ

Быстродействие современных программных систем во многом определяется производительностью сервера СУБД. При этом результирующие характеристики базы данных напрямую зависят от его конфигурации - значений параметров, устанавливаемых специалистом в процессе администрирования. При грамотно выполненной конфигурации сервер СУБД может демонстрировать многократный прирост производительности, измеряемой в количестве выполняемых транзакций в секунду, а также в среднем времени отклика на входящие запросы.

Однако процесс расчета значений параметров требует предварительного анализа следующих факторов:

- Числа пользователей, одновременно выполняющих обращения к системе.
- Аппаратного обеспечения, используемого для размещения сервера СУБД.
- Структуры таблиц, находящихся в базе данных.
- Интенсивности обращения к конкретным данным.
- Изменчивости во времени характера нагрузки на сервер.

Исходя из вышеперечисленного, можно сделать вывод, что процесс администрирования СУБД требует высокой квалификации специалиста, а также может затрачивать большое количество времени в связи с необходимостью в сборе статистических данных.

Проблема заключается в отсутствии поставляемых вместе с СУБД инструментов, позволяющих автоматически производить анализ статистики и вычисление конфигурационных параметров, оптимальных для конкретной инсталляции СУБД, имеющей конкретный профиль нагрузки.

При этом современные сервера СУБД, в том числе PostgreSQL, предоставляют функционал, позволяющий выполнять доступ к различной

статистической информации в автоматическом режиме — используя стороннее программное обеспечение.

Целью данной работы является разработка инструмента динамической и рекомендательной конфигурации сервера БД PostgreSQL, использующего доступные статистические данные. Динамический процесс расчета значений параметров необходим в случаях изменчивости во времени профиля нагрузки.

Объект исследования — анализ статистических данных сервера СУБД PostgreSQL с целью вычисления оптимальных значений конфигурационных параметров.

Предметом исследования является конфигурация параметров сервера СУБД PostgreSQL.

Для достижения цели были поставлены следующие задачи:

- Организация доступа к статистическим данным сервера PostgreSQL.
- Вычисление значений параметров в конкретный момент времени на основе полученных данных.
- Применение полученных значений к конфигурации сервера СУБД как в автоматическом, так и в ручном режиме.
- Автоматические повторные вычисления, выполняемые с задаваемым интервалом.
- Сбор дополнительной необходимой статистики, не предоставляемой сервером СУБД по умолчанию.

ГЛАВА 1. ВНУТРЕННЕЕ УСТРОЙСТВО СЕРВЕРА СУБД

Для достижения поставленных задач необходимо уделить внимание внутреннему устройству СУБД PostgreSQL. Структурно реляционную базу данных можно разделить на два крупных блока:

- Модуль управления физическим хранением данных (таблиц, индексов, материализованных представлений, последовательностей, процедур и так далее).
- Модуль обработки поступающих от приложений-клиентов запросов.

На подсистему физического хранения данных ложится ответственность за сохранность данных - другими словами, за корректное взаимодействие с операционной системой через определенный интерфейс (набор системных функций, например, fsync на операционных системах семейства UNIX).

Модуль обработки SQL-запросов в свою очередь отвечает за преобразования исходного SQL-выражения, полученного в виде строки, в синтаксическое дерево, удобное для последующих операций. После валидации и преобразования начинается стадия построения оптимального плана выполнения, поскольку один и тот же запрос может быть проинтерпретирован несколькими способами.

В качестве примера можно привести таблицу, для некоторого столбца которой построен индекс. В случае поиска по значению в сконфигурированном индексом столбце возможно два исхода: либо индекс будет задействован (будет выполнена операция index scan), либо же индекс будет проигнорирован и таблица будет просканирована последовательно (операция sequential scan). Набор результирующих операций зависит от множества факторов: размера таблицы (количества записей), условия, содержащегося в SQL-запросе, распределения данных в столбце и т.п.

1.1. Описание конфигурации сервера СУБД PostgreSQL

Сервер СУБД PostgreSQL обладает широкими возможностями для конфигурации - в версии 13 для редактирования доступно более 200 параметров.

Параметры имеют следующие типы:

- Логический (допустимые значения - true и false).
- Строковый.
- Целочисленный без единиц измерения.
- Числовой с плавающей точкой без единиц измерения.
- Числовой с единицами измерения.
- Тип перечисления (имеют ограниченный набор допустимых строковых значений).

Изменение значений параметров возможно как через стандартный протокол клиент-сервер (SQL-команды, не входящие в стандарт SQL, но распознаваемые сервером PostgreSQL), так и через файл конфигурации postgresql.conf.

Важной особенностью конфигурации сервера является то, что у каждого параметра имеется так называемый контекст, определяющий то, в какой момент изменение значения способно повлиять на работоспособность сервера. Наиболее распространенными являются:

- User - такие параметры могут быть изменены в рамках одной сессии, а также глобально без необходимости в перезапуске сервера.
- Sighup - изменения таких параметров возможны только глобально, но перезапуск сервера так же не требуется.
- Postmaster - для того, чтобы значения параметров с данным контекстом вступили в силу, требуется перезапуск сервера.
- Superuser - такие параметры могут быть изменены в рамках одной сессии при наличии у пользователя соответствующих прав, а также глобально без необходимости в перезапуске сервера.

1.2. Влияние параметров на производительность сервера

В данном разделе будут рассмотрены группы параметров, влияющих на конкретные аспекты процесса работы сервера СУБД. Основные группы описаны в таблице 1.

Таблица 1. Группы параметров сервера PostgreSQL

Наименование группы	**Описание**
Resource Consumption (потребление ресурсов)	Отвечают за то, каким образом распределяются доступные физические ресурсы между внутренними процессами сервера. Например, какой объем памяти будет выделен под кэш, разделяемый между всеми клиентскими процессами (shared_buffers).
Write Ahead Log (журнал предзаписи)	Определяют поведение процессов, отвечающих за ведение журнала предзаписи, обеспечивающего сохранность данных в случае физического сбоя, а также используемого при репликации данных.
Query Planning (планирование запроса)	Позволяют влиять на построение планов поступающих от клиентских приложений SQL-запросов посредством определения относительных стоимостей различных операций (последовательное сканирование, сканирование индекса и т.д.). Также позволяют конфигурировать генетический оптимизатор запросов (geqo).

Продолжение таблицы 1

Наименование группы	Описание
Automatic Vacuuming (автоматическая очистка)	Значения данных параметров влияют на процессы, связанные с автоматической и ручной очисткой отношений (таблиц и индексов) от записей, подлежащих удалению.
Lock Management (управление блокировками)	Отвечают за поведение сервера при возникновении блокировок. Например, при конкурентном изменении записей несколькими клиентскими процессами, а также при возникновении взаимных блокировок (deadlock).

1.3. Логическая и физическая структуры хранения данных

Логически реляционная база данных состоит из следующих сущностей: таблицы, индексы, представления (в том числе материализованные), последовательности, процедуры.

Способ же хранения перечисленных выше структур данных зависит от реализации конкретной СУБД. PostgreSQL хранит таблицы, материализованные представления и индексы в виде файлов размером до 1GB. В случае превышения данного объема сущность разбивается на несколько файлов, каждый из которых также не превышает 1GB. При этом логически структура данных остается целостной.

Каждый такой файл разбит на сегменты, именуемые страницами. По умолчанию размер каждой страницы равен 8KB. Изменение размера страницы возможно только при перекомпиляции исходных кодов СУБД.

На рисунке 1 приведено графическое представление структуры страницы.

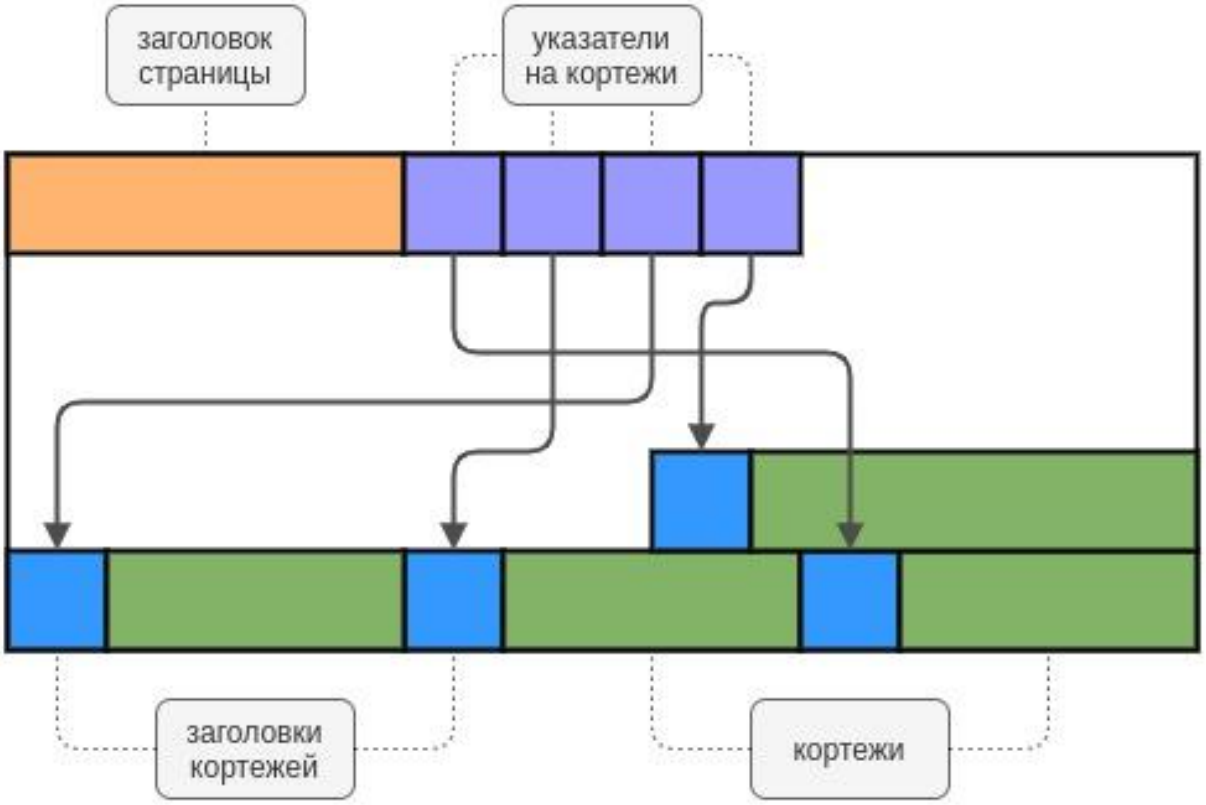

Рисунок. 1. Структура страницы данных в PostgreSQL

Заголовок страницы содержит такие данные, как контрольная сумма, различные системные флаги, а также смещения в байтах относительно начала страницы для последнего указателя на кортеж.

Заголовок записи хранит различные метаданные кортежа, наиболее важными из которых являются поля t_xmin и t_xmax, обозначающие идентификаторы транзакций в рамках которых было выполнено добавление и удаление кортежа соответственно.

1.4. Процесс обработки SQL-запроса

Процесс обработки отправленного клиентом SQL-запроса состоит из нескольких этапов:

1. Получение запроса в виде строки.
2. Грамматический разбор запроса. В случае, если запрос содержит синтаксические ошибки, на данном этапе клиенту будет возвращено сообщение с описанием ошибки. Если запрос синтаксически корректен, в результате формируется дерево разбора выражения.
3. Построение всех возможных планов запроса на основании полученного на предыдущем этапе дерева. Каждый план также представляет собой синтаксическое дерево, однако, в отличие от дерева разбора, содержит информацию о деталях выполнения операций. Например, то, каким образом будет выполнено объединение двух отношений. При этом каждая операция имеет относительную стоимость.
4. Выбор наиболее оптимального плана запроса на основании общей стоимости всех операций в плане.
5. Выполнение запроса с использованием наиболее оптимального плана.
6. Формирование результирующей коллекции кортежей, передаваемой по сети клиентскому приложению.

1.5. Метаданные сервера

СУБД PostgreSQL предоставляет доступ к большому количеству данных о структуре таблиц, их связях, типах столбцов, наличию индексов и т.д. посредством системных каталогов - таблиц и представлений в схеме pg_catalog. Поскольку основной задачей данной работы является достижение частичной автоматизации процесса администрирования СУБД, метаданные сервера играют особенно важную роль.

Все системные каталоги в PostgreSQL представляют собой стандартные структуры данных, отличающиеся от создаваемых пользователем лишь

частичным ограничением прямой модификации (через операторы DML - insert, update, delete).

Помимо информации о статической структуре отношений, системные каталоги также содержат статистические данные о внутренних процессах (таких как autovacuum - автоматическая очистка и autoanalyze - автоматическое построение гистограмм на основе данных в пользовательских таблицах).

Ниже приведена таблица 2, содержащая описание системных каталогов.

Таблица 2. Системные каталоги в PostgreSQL

Наименование и тип каталога	**Описание**
pg_class, таблица	Содержит информацию обо всех сущностях, которые на физическом уровне представление в виде множества страниц, размер которых по умолчанию равен 8KB.
pg_inherits, таблица	Предоставляет данные о родительских связях между таблицами в базе. Может служить источником данных при анализе наследуемых и партиционированных таблиц.
pg_indexes, представление	Содержит данные обо всех индексах.
pg_settings, представление	Является источником информации о текущей конфигурации сервера.
pg_stat_all_tables, представление	Содержит одну запись для каждой таблицы в базе со статистическими данными о выполняемых по отношению к ней операциям.
pg_stat_activity, представление	Содержит информацию о текущих процессах сервера. Может служить источником данных о выполняемых на данный момент запросах, а также о системных процессах, таких как vacuum, autovacuum, pg_dump.

ГЛАВА 2. СУЩЕСТВУЮЩИЕ РЕШЕНИЯ

Для достижения качественного результата в рамках данной работы необходимо провести изучение и детальное сравнение существующих инструментов, направленных на вычисление оптимальных значений параметров конфигурации сервера PostgreSQL.

2.1. Описание инструмента Pgtune

Данный инструмент предоставляет WEB-интерфейс для ручного внесения параметров СУБД и получения значений параметров. В качестве входных параметров pgtune принимает:

- Версия Postgres (доступны для выбора 9.2 - 12).
- Тип операционной системы (Linux, MacOS, Windows).
- Тип приложения, со стороны которого происходит обращение к базе (веб-приложение, OLTP-система, настольное приложение).
- Объем оперативной памяти сервера в гигабайтах или мегабайтах.
- Число физических ядер сервера - не обязательно для заполнения.
- Максимальное число одновременно работающих клиентов БД - не обязательно для заполнения.
- Тип диска (HDD или SSD).

На основе данных, введенных пользователем, производится вычисление значений следующих параметров: max_connections, effective_cache_size, maintenance_work_mem, checkpoint_completion_target, wal_buffers, default_statistics_target, random_page_cost, effective_io_concurrency, max_worker_processes, max_parallel_workers, max_parallel_workers_per_gather, max_parallel_maintenance_workers, work_mem, min_wal_size, max_wal_size.

Таким образом, данный инструмент не взаимодействует с сервером СУБД, и не дает возможности автоматического пересчета значений. Однако в простых сценариях, когда к производительности сервера не накладываются высокие требования, применение данного решения может оказаться оптимальным выбором ввиду его простоты и отсутствия необходимости во встраивании нового инструмента в существующую инфраструктуру программного комплекса.

2.2. Описание инструмента Postgres-checkup

Postgres-checkup - это решение для глубокой диагностики сервера СУБД PostgreSQL. Он автоматически определяет текущие и потенциально возможные проблемы, связанные с производительностью, масштабируемостью и безопасностью. Также данный инструмент генерирует рекомендации в текстовом виде по устранению выявленных проблем. Разработчики рекомендуют выполнять анализ на регулярной основе, а также до и после крупных изменений логической и/или физической структуры БД.

Основная особенность данного инструмента заключается в том, что он производит анализ широкого спектра данных - информации из системных каталогов сервера (описание каталогов приведено в разделе 1.5 Метаданные сервера), данных о конфигурации сервера и т.д. Также он производит комплексный анализ распределенного кластера, содержащего мастер-сервер и реплики.

Все отчеты, генерируемые postgres-checkup, можно разделить на две группы:

- Отчеты в формате JSON, рассчитанные на потребление внешними программными решениями.

- Отчеты в формате Markdown, предоставляющие всю информацию в удобном для восприятия человеком виде.

Основной недостаток данного решения заключается в том, что он предоставляет лишь рекомендации для администратора СУБД, однако не производит вычисления значений параметров. Но ввиду большого количества отчетов, формируемых в удобном формате, postgres-checkup является отличным инструментом для поиска проблем и комплексного анализа сервера.

2.3. Описание инструмента Pgconfig 2.0

Данный инструмент во многом дублирует функциональность рассмотренного выше Pgtune. По аналогии с ним Pgconfig 2.0 предоставляет веб-интерфейс для внесения информации о физической конфигурации сервера (число ядер ЦПУ, объем оперативной памяти, тип дисков), среднем числе активных пользователей, профиле нагрузки. Однако по сравнению с Pgtune данное решение имеет несколько недостатков:

- На момент написания данной работы находится в стадии бета-тестирования. Другими словами, может быть непригодно для промышленного использования.
- Заявлена поддержка PostgreSQL не выше версии 10.

2.4. Сравнение возможностей аналогов

Ниже приведена таблица 3, отражающая основные функциональные особенности рассмотренных выше инструментов, а также инструмента,

разрабатываемого в рамках данной работы. Знаком “+” в ячейках отмечены единицы функционала, доступные в конкретном инструменте.

Таблица 3. Сравнение функционала инструментов администрирования СУБД

Функционал	**Pgtune**	**Postgres checkup**	**Pgconfig 2.0**	**Разраб-мый инструмент**
Вычисление значений параметров конфигурации	+		+	+
Анализ данных из системных каталогов СУБД		+		+
Доступ к информации о физической конфигурации сервера		+		+
Автоматический периодический анализ СУБД				+
Предоставление общих рекомендаций для администратора СУБД		+		
Хранение дополнительной статистики в отдельном хранилище				+
Наличие графического интерфейса	+		+	+
Автоматическое конфигурирование сервера СУБД				+

Исходя из проведенного анализа можно сделать вывод, что существующие на данный момент решения обладают рядом недостатков:

- Не реализуют периодический анализ СУБД в автоматическом режиме, что крайне важно при изменчивости во времени характера нагрузки на сервер.

- Не предоставляют возможностей по автоматизации конфигурирования сервера.

ГЛАВА 3. ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

Данная глава посвящена рассмотрению технологию, используемых при создании инструмента конфигурации сервера СУБД.

3.1. Технологии, используемые при разработке инструмента

Инструмент реализован на платформе .NET с использованием языка программирования C#. Достигнута совместимость инструмента как с операционной системой Windows 10, так и с UNIX-подобными операционными системами (Ubuntu, Debian, CentOS), что является важным фактором в связи с потенциальной разнородностью промышленной инфраструктуры.

Для взаимодействия с сервером СУБД посредством стандартного протокола клиент-сервер использована библиотека Npgsql. В качестве инструмента построения графического интерфейса используется библиотека Terminal.Gui.

Взаимодействие с инструментом построено посредством командной строки и отрисовываемого в терминале графического интерфейса. Данный выбор обоснован тем, что разрабатываемый инструмент рассчитан на функционирование в одном сетевом контуре с СУБД, что накладывает ограничения на доступ извне.

3.2. Технологии, используемые при тестировании производительности сервера СУБД

Для определения корректности и оптимальности значений параметров, вычисляемых разрабатываемым инструментом, необходимо провести комплексное тестирование производительности сервера СУБД до и после применения инструмента.

В качестве платформы для размещения серверов СУБД и инструмента использованы облачные сервисы AWS. Для генерации нагрузки на сервер БД использован инструмент Pgbench, входящий в комплект поставки PostgreSQL.

ГЛАВА 4. РАЗРАБАТЫВАЕМЫЙ ИНСТРУМЕНТ

Данный раздел посвящен детальному описанию разрабатываемого инструмента, его внутренних процессов, архитектуры, функциональных возможностей.

4.1. Описание инструмента

Инструмент динамической и рекомендательной конфигурации сервера PostgreSQL состоит из следующих компонентов:

- Центральное, так называемое хостовое приложение, содержащее основную логику, связанную с:
    - Вычислением значений параметров.
    - Автоматическим запуском периодических процессов.
    - Доступом к системному хранилищу данных.
    - Взаимодействием с серверами СУБД.

  Хостовое приложение находится в запущенном состоянии на протяжении долгого времени, т.к. несет ответственность за вычисления и сбор статистики во времени.
- Приложение-агент, функционирующее на физических или виртуальных серверах, на которых развернуты севера PostgreSQL. Агент предоставляет HTTP API, используемое хостовым приложением для получения информации о физической конфигурации сервера - о количестве ядер ЦПУ, общем объеме оперативной памяти.

  Данное приложение так же должно постоянно находится в запущенном состоянии, т.к. определенная часть логики в основном приложении опирается на данные, получаемые от агента.

- Приложение управления, предоставляющее возможность контроля основным приложением посредством графического интерфейса. Также приложение управления обеспечивает доступ к результирующим данным, т.е. к значениям параметров, вычисленных центральным приложением. Доступ к значениям необходим в случае отключения автоматического применения вычислений к конфигурации сервера, т.е. в случае, когда инструмент функционирует исключительно в рекомендательном режиме.

Ниже приведена диаграмма 2, демонстрирующая высокоуровневую архитектуру разрабатываемого инструмента.

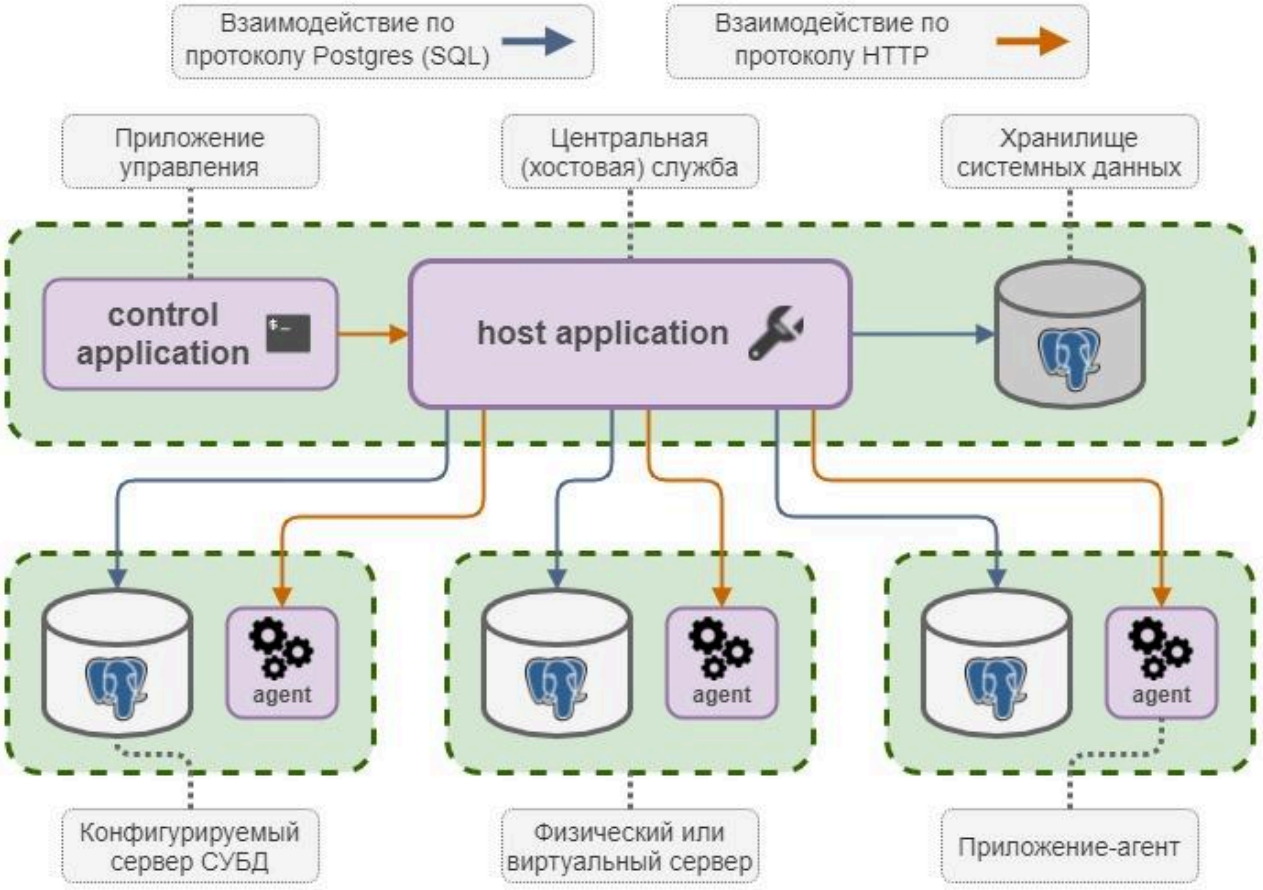

Рисунок 2. Высокоуровневая архитектура

На диаграмме отражены программные компоненты инструмента, описанные выше - центральная (хостовая) служба, приложение-агент и

приложение управления. Помимо модулей инструмента на схеме присутствуют хранилище данных и конфигурируемые сервера СУБД. В данном случае control application, host application и system storage размещены на одном сервере, однако с технической точки зрения такого ограничения нет, и все компоненты могут быть размещены на отдельных серверах.

При использовании инструмента к инфраструктуре предъявляются следующие требования:

1. Должна быть налажена сетевая связность между серверами согласно приведенной выше диаграмме.
2. Приложения-агенты должны быть размещены строго на тех же серверах, что и СУБД. В противном случае значения вычисленных параметров могут оказаться некорректными, т.к. будут основаны на некорректных данных.

4.2. Детализация архитектуры инструмента

Далее будет приведено детальное описание отдельных модулей и процессов, входящих в состав центральной (хостовой) службы.

Инструмент реализован с использованием компонентной архитектуры. Другими словами, кодовая база разделена на обособленные модули, каждый из которых предоставляет определенный публичный контракт, используемый другим компонентом или компонентами.

Ниже представлена диаграмма 2, отражающая структуру компонентов центральной службы инструмента.

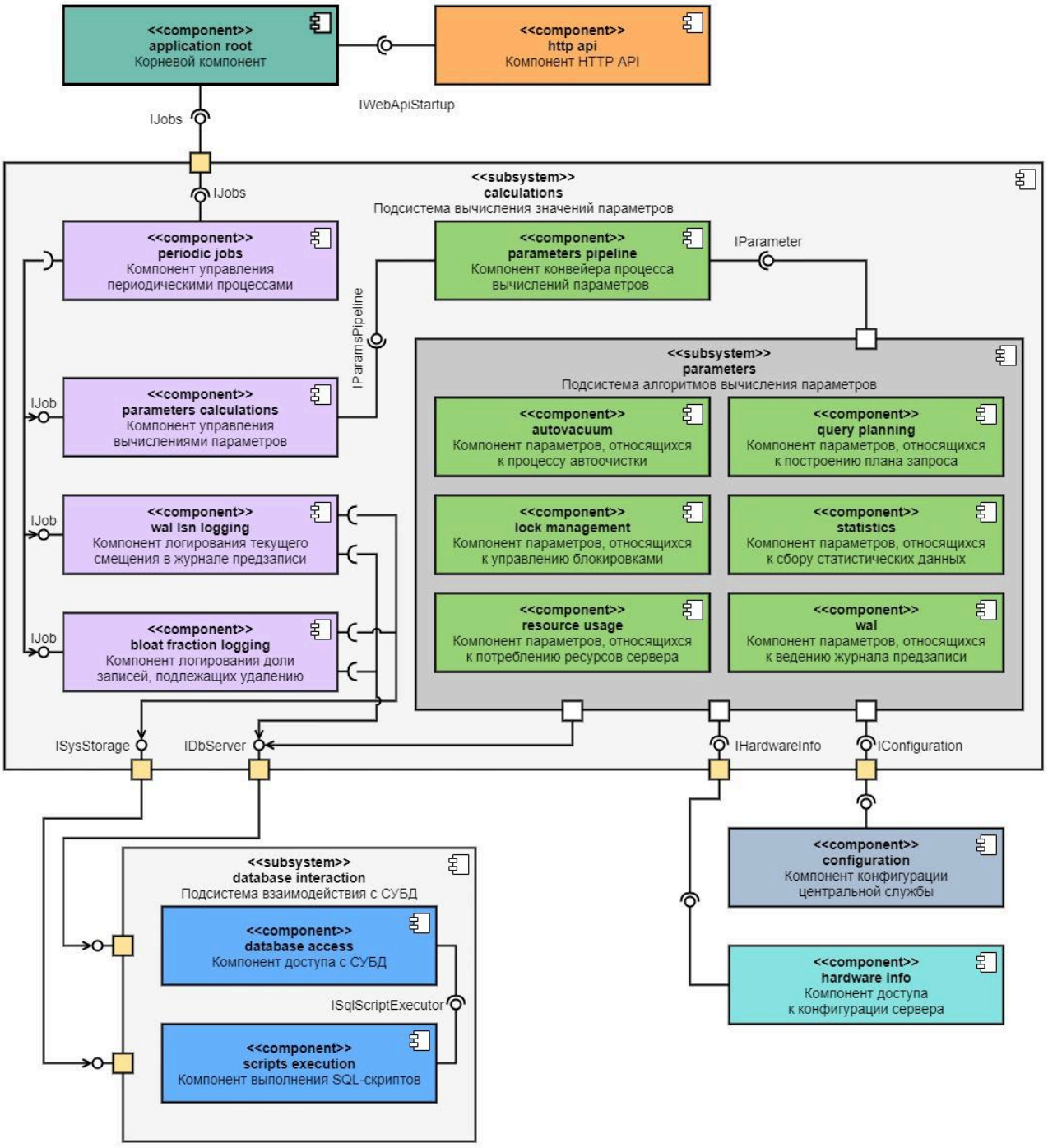

Рисунок 3. Диаграмма компонентов центральной службы

Таким образом, структурно приложение представляет собой ациклический ориентированный граф, узлами которого являются компоненты.

Задача корневого компонента состоит в инициализации приложения, состоящем из нескольких этапов:

1. Чтение файла конфигурации.
2. Запуск программного HTTP-интерфейса.

3. Инициализация всех подключенных компонентов.
4. Запуск периодических процессов.

После завершения инициализации хостовое приложение начинает обрабатывать запросы, поступающие от управляющего приложения.

Ответственность компонента управления периодическими процессами заключается в своевременном запуске итераций согласно значениям, указанным в файле конфигурации. Все процессы выполняются параллельно и независимо друг от друга, что дает лучшую производительность по сравнению с синхронным запуском.

Компонент управления вычислениями параметров запускает конвейер вычислений, который в свою очередь содержит несколько шагов:

1. Построение ациклического направленного графа, в вершинах которого находятся объектные представления параметров, типы которых содержат логику вычисления значений. Необходимость построения графа вызвана тем, что вычисления некоторых параметров опираются на значения других параметров. Соответственно, параметры, являющиеся зависимостями текущего параметра, должны быть рассчитаны до запуска вычисления его значения.
2. Обход построенного графа, начиная с узлов, не имеющих зависимостей. При этом полученные значения сохраняются в буфере на уровне оперативной памяти приложения.
3. Добавление полученных значений в системное хранилище для сохранения истории изменений по всем параметрам. Является необходимым шагом в случае, если инструмент функционирует в рекомендательном режиме.
4. Применение полученного набора значений к конфигурации сервера. Исходя из того, что вычисления являются зависимыми друг от друга, изменение конфигурации сервера СУБД должно выполняться атомарно. В противном случае возможно неполное применение значений, что в свою очередь может негативно отразиться на производительности СУБД.

Компонент логирования текущего смещения в журнале предзаписи отвечает за создание временного ряда значений в байтах, отражающих объем трафика, обработанного внутренними механизмами PostgreSQL. Далее полученные значения используются при вычислении параметров, связанных с WAL-процессами. Более подробное описание будет приведено далее. С технической точки зрения смещение хранится в виде пары 64-х битных беззнаковых чисел. Получение текущего значение осущетвляется путем вызова системной функции pg_current_wal_insert_lsn, находящейся в схеме pg_catalog.

Компонент логирования доли записей, подлежащих удалению выполняет сбор статистических данных о процентном соотношении активных и подлежащих удалению кортежей среди всех таблиц в базе данных. Значения формируются на основе данных в системном представлении pg_catalog.pg_stat_all_tables (столбцы n_dead_tup и n_live_tup). Полученный временной ряд используется для вычислений параметров, связанных с внутренними процессами автоочистки. Об этом также будет сказано более подробно в следующих разделах.

Подсистема алгоритмов вычисления параметров содержит в себе основную логику инструмента - функции, возвращаемыми значениями которых являются значения параметров, применяемые к конфигурации СУБД. Данная подсистема разделена на шесть компонентов:

- Autovacuum - компонент параметров, относящихся к процессу автоочистки.
- Lock management - компонент параметров, относящихся к управлению блокировками.
- Resource usage - компонент параметров, относящихся к потреблению ресурсов сервера (мощностей центрального процессора и оперативной памяти).
- Query planning - компонент параметров, относящихся к построению планов поступающих от приложений-клиентов запросов.

- Statistics - компонент параметров, влияющих на внутренние процессы PostgreSQL, связанные с формированием статистических данных.
- Write ahead log - компонент, содержащий параметры, относящиеся к ведению журнала предварительной записи сервером PostgreSQL.

Компонент доступа к конфигурации сервера содержит логику взаимодействия с приложением-агентом, который в свою очередь получает данные количестве ядер центрального процессора и о количестве оперативной памяти посредством вызова системных функций.

Внутреннее устройство приложения-агента описано при помощи диаграммы классов (рисунок 4).

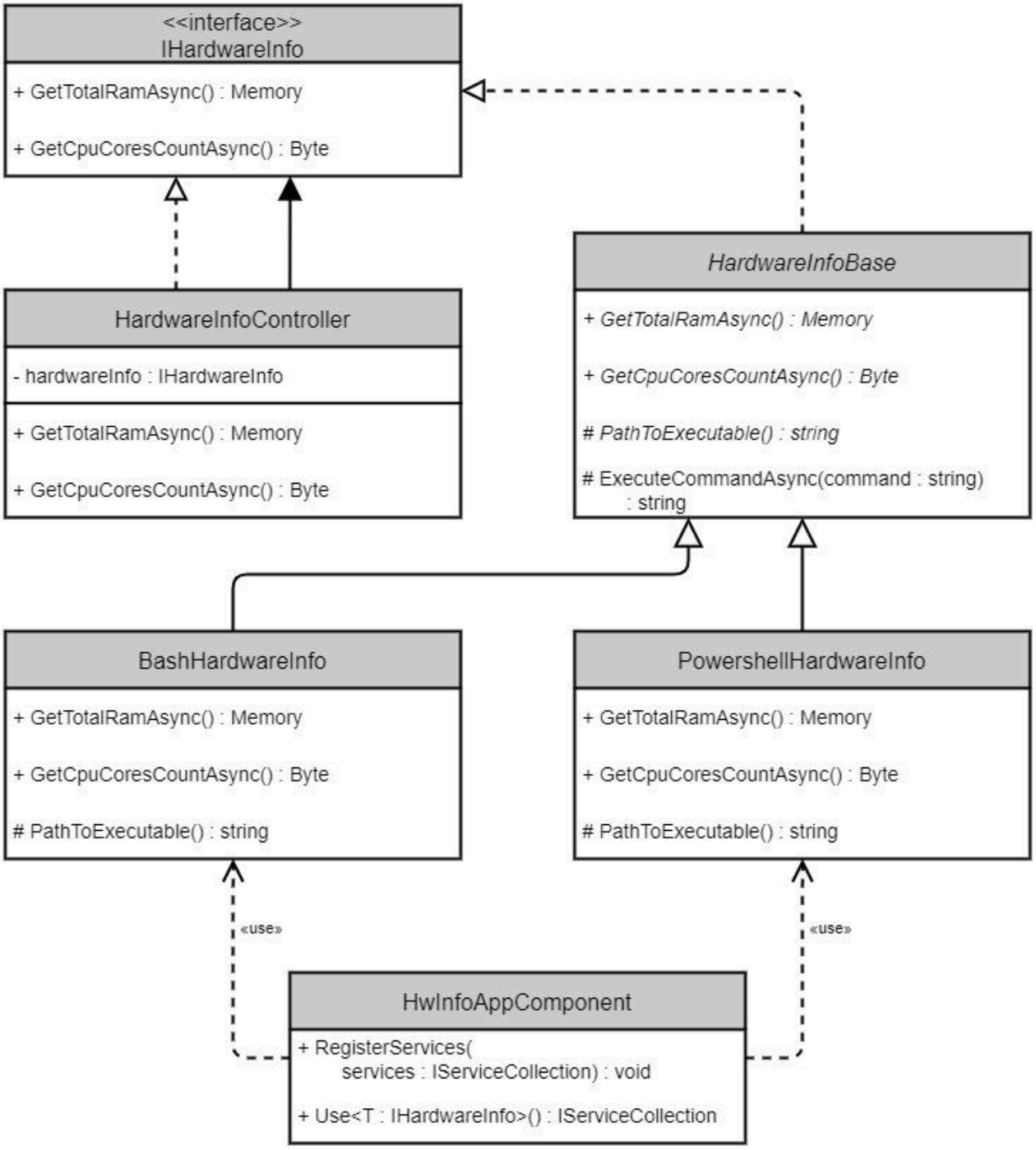

Рисунок 4. Диаграмма классов приложения-агента

Основным элементом в структуре типов приложения-агента является интерфейс IHardwareInfo, декларирующий методы для получения информации о ЦПУ и оперативной памяти. Данный интерфейс имеет три реализации:

- Тип HardwareInfoController, задача которого заключается в перенаправлении поступающих от центральной службы HTTP-запросов к фактической реализации данного интерфейса.
- Тип BashHardwareInfo, используемый при работе агента на операционных системах семейства Linux.
- Тип PowershellHardwareInfo, функционирующий при запуске агента на Windows.

Выбор между BashHardwareInfo и PowershellHardwareInfo осуществляется автоматически при запуске агента. Таким образом достигается универсальность инструмента.

Далее следует вернуться к рассмотрению устройства центральной службы и взять во внимание подсистему взаимодействия с сервером СУБД.

Данный модуль состоит из двух компонентов:

- Компонент выполнения SQL-скриптов, отвечающие за поддержание системного хранилища данных в консистентном состоянии. При первом подключении к системному хранилищу выполняется проверка актуальности схемы. В случае, если структура таблиц и типов в хранилище не актуальна, выполняется чтение SQL-скриптов из внутренних ресурсов приложения. Далее все скрипты последовательно выполняются, и инструмент переходит в стандартный режим работы.
- Компонент доступа с СУБД, взаимодействующий с сервером PostgreSQL посредством SQL-запросов - как с конфигурируемыми серверами, так и с системным хранилищем данных.

4.3. Структура системного хранилища данных

Данный раздел содержит описание структуры системного хранилища данных. В качестве хранилища используется сервер PostgreSQL.

Все таблицы, типы и функции системного хранилища, реализованного с использованием PostgreSQL, располагаются в отдельной схеме, именуемой “marula_tool”. Ниже представлена диаграмма “сущность-модель” (рисунок 5), описывающая структуру в графическом виде.

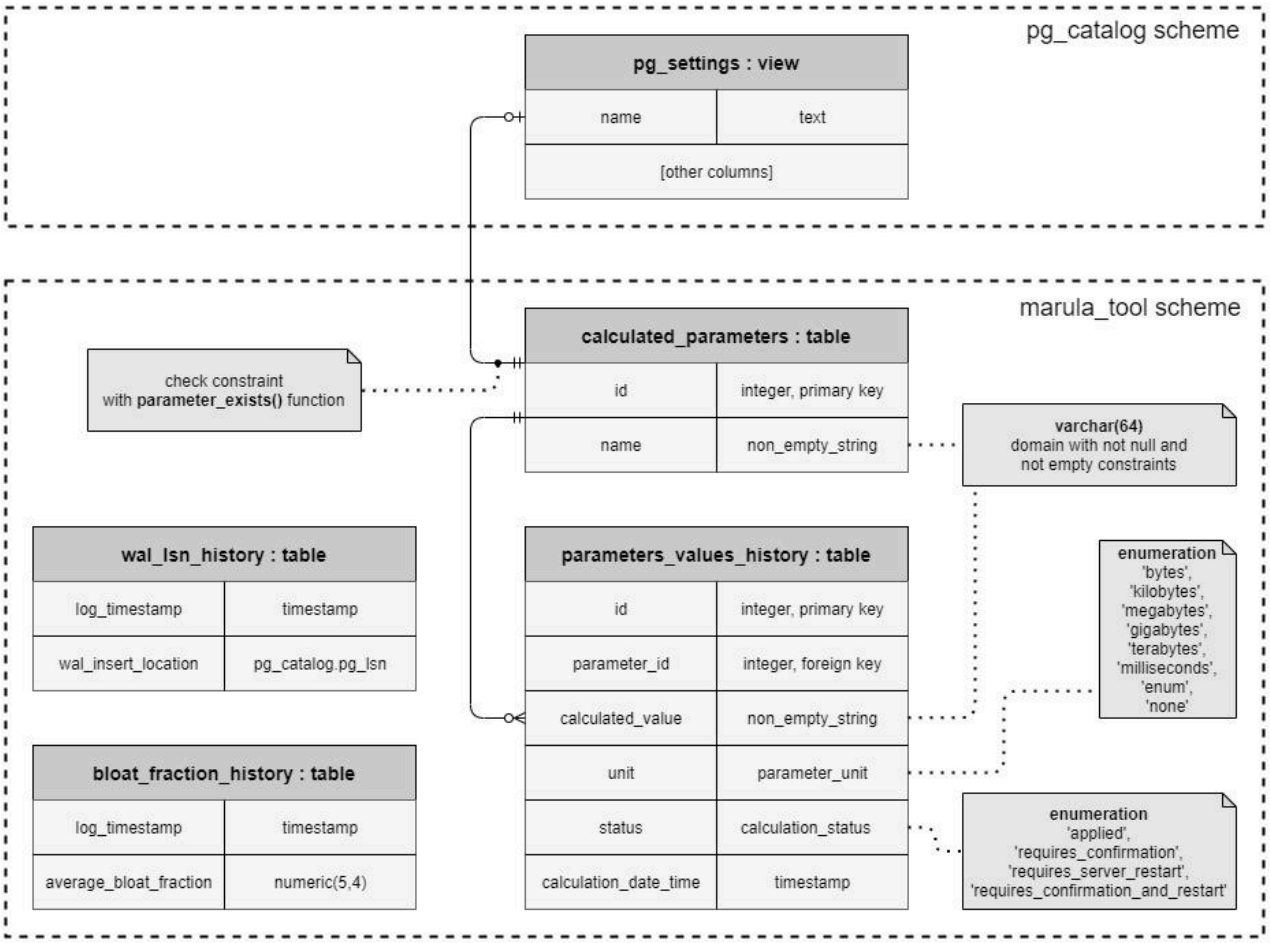

Рисунок 5. Диаграмма структуры системного хранилища данных инструмента

Таблица calculated_parameters представляет собой справочник параметров сервера СУБД, заполняемый именами параметров, вычисляемых и конфигурируемых центральной службой. Для сохранности консистентности данных столбец name сконфигурирован ограничением целостности, проверяющем существование параметра в конфигурации сервера СУБД на основе данных в системном представлении pg_catalog.pg_settings, указанном в таблице 2.

Таблица parameters_values_history содержит все значения параметров, вычисленные до текущего момента центральной службой. Данная таблица имеет столбец parameter_id, являющийся внешним ключом, ссылающимся на стоблец id таблицы calculated_parameters. Стоблец calculated_value непосредственно хранит вычисленные значения. Строковый тип выбран как наиболее универсальный вариант, подходящий для хранения значений различных типов. Стоблец unit содержит единицы измерения значений и представлен типом parameter_unit, декларированном на уровне структуры БД. Столбец calculation_status хранит информацию о вычислении. Также представлен отдельным типом на уровне БД и может принимать следующие значения:

- Applied - в случае, если инструмент функционирует в режиме автоматического применения значений, и данный параметр доступен для изменения без перезапуска сервера СУБД.
- Requires confirmation - присваивается вычисленному значению в рамках рекомендательного режима работы. При этом параметр может быть изменен без перезапуска сервера.
- Requires server restart - означает, что инструмент находится в режиме автоматического применения значений, значение параметра на уровне конфигурации СУБД было изменено посредством команды alter system, но требуется перезапуск сервера.
- Requires confirmation and restart - присваивается в случае, когда отключено автоматическое применение значений и для вступления в силу примененного к конфигурации значения требуется перезапуск СУБД.

Таблица wal_lsn_history содержит историю байтового смещения в журнале предзаписи, отражающего объем трафика, обработанного внутренними механизмами WAL PostgreSQL. Данные из этой таблицы используются при вычислении значений параметров, связанных с журналом пред. записи.

Таблица bloat_fraction_history хранит временной ряд доли записей, подлежащих удалению. Так же, как и таблица wal_lsn_history,

bloat_fraction_history заполняется периодическими процессами, описанными в разделе 4.2.

4.4. Конфигурация инструмента

Все модули инструмента (центральная служба, приложение-агент и приложение управления) имеют возможность конфигурирования посредством файлов в формате JSON.

Центральная служба конфигурируется при помощи файла marula-host-config.json и содержит параметры, описанные в таблице 4.

Таблица 4. Параметры конфигурации центральной службы инструмента

Имя параметра	**Тип**	**Описание**
ConnectionString	Строка	Строка подключения к конфигурируемому серверу СУБД.
RecalculationInterval	Число	Интервал в секундах, с которым запускается периодический процесс вычисления значений параметров.
AutoAdjustParams	Булево	Режим работы центральной службы. Если присвоено значение false, то инструмент функционирует в рекомендательном режиме. В противном случае на каждой итерации процесса вычислений происходит модификация конфигурации СУБД.

Продолжение таблицы 4

Имя параметра	**Тип**	**Описание**
AgentApiUri	Строка	Сетевой адрес, по которому выполняются обращения к API приложения-агента.
BloatLoggingInterval	Число	Интервал в секундах, с которым запускается периодический процесс логирования доли записей, подлежащих удалению.
RollingWindow (bloat)	Число	Размер временного окна в секундах, определяющий объем данных, участвующих при вычислении скользящего среднего числа доли записей, подлежащих удалению. Подробнее данный процесс описан в разделе 4.5.
LsnLoggingInterval	Число	Интервал в секундах, с которым запускается периодический процесс логирования текущего смещения в журнале предзаписи.
RollingWindow (WAL)	Число	Размер временного окна в секундах, определяющий объем данных, участвующих при вычислении объема трафика, попадающего в журнал предварительной записи. Подробнее данный процесс описан в разделе 4.5.

4.5. Вычисление значений параметров

Данный раздел посвящен описанию формул и алгоритмов, используемых при вычислении параметров центральной службой инструмента.

4.5.1. Параметры, связанные с процессами автоочистки

Далее будут рассмотрены внутренние алгоритмы инструмента, направленные на конфигурирование процесса автоочистки в PostgreSQL.

Параметр autovacuum активирует или дезактивирует функционирование процесса автоочистки. Ввиду того, что инструмент выполняет вычисления в контексте автоматической очистки, этот параметр всегда устанавливается в значение true. В формализованном виде присвоение представлено формулой (1).

$$autovacuum = true \quad (1)$$

Параметр autovacuum_vacuum_scale_factor определяет процентное соотношение записей, подлежащих удалению, при котором таблица определяется как требующая очистки. Значение по умолчанию равно 0.2. То есть когда в таблице образуется 20% ‘неактивных’ записей, эта таблица будет очищена процессом автоочистки. Для вычисления оптимального значения в инструменте используется следующая формула:

$$autovacuum\_vacuum\_scale\_factor = min(0.2,\ 10^4 / average\_table\_size), \quad (2)$$

где average_table_size - средний размер таблицы в БД.

Значение average_table_size вычисляется на основе данных в системном представлении pg_catalog.pg_stat_user_tables. При этом пустые таблицы не учитываются.

Параметр autovacuum_vacuum_cost_delay используется для конфигурации временной задержки, выполняемой процессом автоочистки для снижения влияния операции VACUUM на общую производительность СУБД. Значение по умолчанию в версии PostgreSQL 12 и более новых равно 2 мс. В более ранних версиях это значение равно 20 мс, что может оказаться слишком большим значением, учитывая производительность современных физических и виртуальных серверов. Значение данного параметра снижается для PostgreSQL 11 и более ранних версий, устанавливается равным 2 мс.

$$autovacuum = 2ms$$
(3)

Параметр autovacuum_vacuum_cost_limit определяет лимит по числу операций, выполняемых процессом автоочистки за одну итерацию. После достижения данного лимита автоматическая очистка прекращается на временной интервал, равный значению параметра autovacuum_vacuum_cost_delay. Вычисление значения основано на данных, полученных в результате работы периодического процесса логирования доли записей, подлежащих удалению. Реализация получения среднего числа таких записей представлена в приложении А. В таблице 5 показан пример выборки из таблицы bloat_fraction_history системного хранилища инструмента.

Таблица 5. Пример выборки данных из таблицы bloat_fraction_history

Дата и время сохранения записи	**Среднее значение**
2021-06-01 0:00:00	0,2
2021-06-01 3:00:00	0,4

2021-06-01 6:00:00	0,3

Продолжение таблицы 5

Дата и время сохранения записи	Среднее значение
2021-06-01 9:00:00	0,35
2021-06-01 12:00:00	0,5
2021-06-01 15:00:00	0,4
2021-06-01 18:00:00	0,6
2021-06-01 21:00:00	0,7

Данные, приведенные в таблице 5, представлены в графическом виде на рисунке 6.

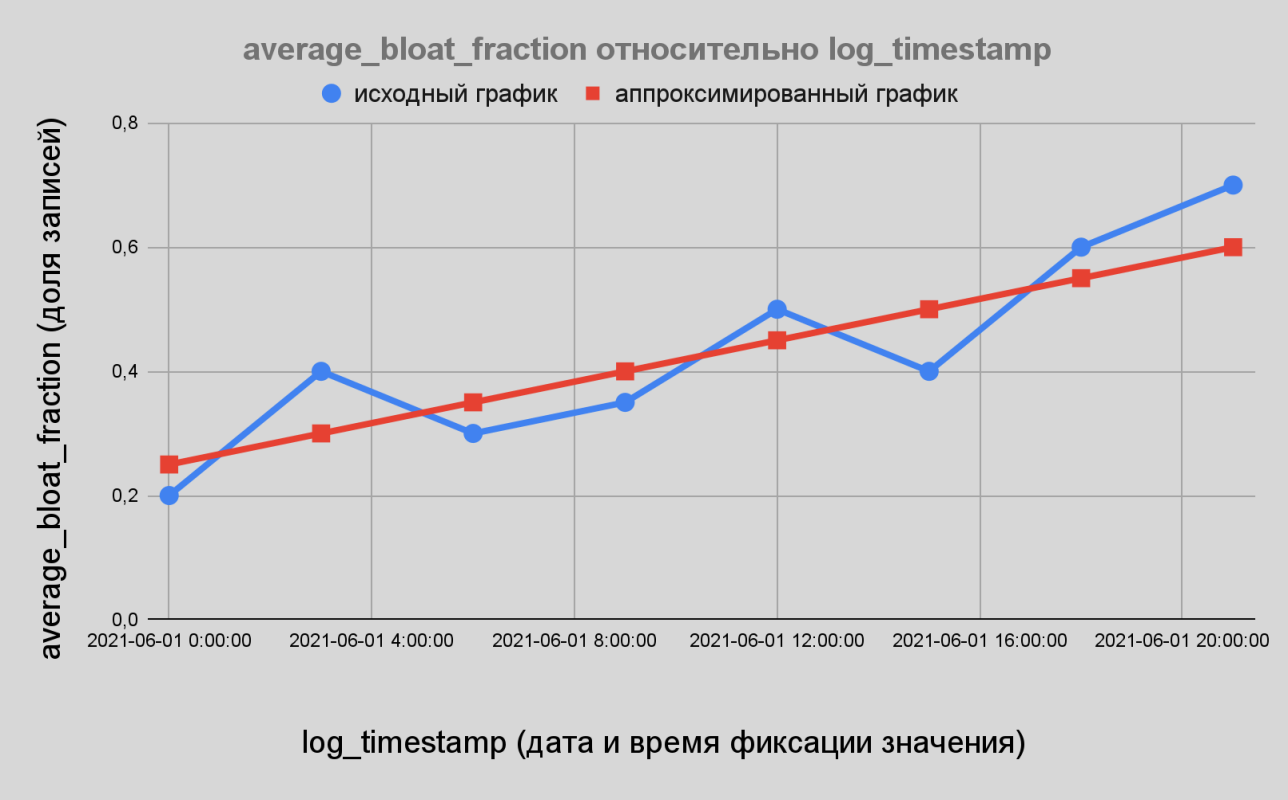

Рисунок 6. Пример построения графика среднего значения доли записей, подлежащих удалению

Для дальнейших вычислений выполняется преобразование исходной функции путем линейной регрессии. Используется метод наименьших квадратов. График полученной линейной функции изображен на рисунке 6.

Далее происходит извлечение следующих значений:

- trend_coefficient - первая производная линейной функции, тангенс угла наклона линии.
- bloat_constant - свободный член линейной функции.

Полученные числовые коэффициенты используются для вычисления значения параметра autovacuum_vacuum_cost_limit. Исходя из значения trend_coefficient можно сделать вывод о том, растет или уменьшается со временем среднее значение записей, подлежащих удалению. Если значение растет (производная функции положительна), значение autovacuum_vacuum_cost_limit должно быть увеличено относительно текущего значения. Если доля средних записей уменьшается с повышенным темпом, то значение параметра соответственно уменьшается относительно текущего значения. Вычисление выполняется по формуле (4).

$$autovacuum\_vacuum\_cost\_limit = current\_value \times (1 + max(-0.2, min(0.2, trend\_coefficient)), \quad (4)$$

где current_value - текущее значение параметра autovacuum_vacuum_cost_limit, trend_coefficient - производная линейной функции.

Однако возможна ситуация, в которой на момент запуска вычисления значения параметра autovacuum_vacuum_cost_limit в системном хранилище отсутствует достаточное количество данных для построения аппроксимированной функции (необходимо как минимум две записи). В таком случае вычисление производится по упрощенной формуле:

$$autovacuum\_vacuum\_cost\_limit = 500 \times autovacuum\_max\_workers, \quad (5)$$

где autovacuum_max_workers - значение одноименного параметра.

Параметр autovacuum_naptime задает минимальный временной интервал между двумя запусками процесса автоочистки. Значение по умолчанию равно 60 секунд. Для повышения производительности автоочистки данному параметру присваивается значение 30 секунд.

$$autovacuum\_naptime \ = \ 30s$$
(6)

Параметр autovacuum_max_workers определяет степень параллелизма процесса автоматической очистки. Значение по умолчанию равно трем. Вычисление производится инструментом на основе данных о конфигурации сервера, полученных от приложения-агента, и приведено в формуле 7.

$$autovacuum\_max\_workers \ = \ 0.5 \ \times \ number\_of\_cpu\_cores,$$
(7)

где number_of_cpu_cores - число процессорных ядер на сервере СУБД.

Параметр autovacuum_vacuum_threshold задаёт минимальное число измененных или удаленных записей, при котором в процессе автоматической очистки будет выполняться VACUUM для отдельно взятой таблицы. Значение по умолчанию - 50. Вычисление данного параметра основывается на среднем размере таблицы в базе данных и выполняется согласно формуле 8.

$$autovacuum\_vacuum\_threshold \ = \ max(50, \ 0.01 \ \times \ average\_table\_size),$$
(8)

где average_table_size - средний размер таблицы в БД.

4.5.2. Параметры, связанные с управлением блокировками

Параметр max_locks_per_transaction определяет максимально допустимое число блокировок для каждой отдельной транзакции в базе данных. Имеет контекст postmaster, соответственно для вступления в силу примененного к конфигурации значения требуется перезапуск сервера. Вычисление основано на данных о родительских связях между таблицами (в рамках декларативного партиционирования или прямого наследования), т.к. возможны ситуации, в которых модификация родительской таблицы приводит к блокировкам дочерних таблиц, и при большом числе партиций значения по умолчанию может быть недостаточно. Однако если текущее значение параметра превышает вычисленное значение, параметр не модифицируется, т.к. оно могло быть изменено вручную администратором СУБД, опираясь на аспекты, не связанные с партиционированием таблиц. Ниже приведена формула (9), отражающая вычисление.

$$max\_locks\_per\_transaction = max(1.2 \times max\_parts, current\_value), \quad (9)$$

где current_value - текущее значение параметра max_locks_per_transaction, max_parts - максимальное число партиций среди всех таблиц в базе данных.

Каскадное партиционирование (когда партиции родительской таблицы также имеют свои партиции) учитывается при вычислении max_partitions_count. Реализация получения иерархических связей представлено в приложении Б данной работы.

4.5.3. Параметры, связанные с потреблением ресурсов

Параметр shared_buffers определяет объем оперативной памяти, доступной серверу СУБД для хранения кэшированных данных. Имеет контекст postmaster, требует перезапуска сервера для применения значения. Значение по умолчанию равно 128 МБ. Вычисление основано на данных, полученных от приложения-агента, и описано в формуле (10).

$$shared\_buffers = 0.25 \times total\_ram\_size,$$

(10)

где total_ram_size - общий объем памяти на сервере СУБД

Параметр work_mem устанавливает объем памяти, доступной для операций сортировки, хэш-объединения в рамках каждой сессии. Вычисление основано на данных, полученных от приложения-агента, а также на максимально возможном числе параллельно активных клиентов, задаваемом параметром max_connections.

$$work\_mem = 0.25 \times total\_ram\_size \div max\_connections,$$

(11)

где total_ram_size - общий объем памяти на сервере СУБД, max_connections - текущее значение одноименного параметра.

Параметр maintenance_work_mem определяет максимальный объем памяти, потребляемой такими операциями как VACUUM, CREATE INDEX, и ALTER TABLE ADD FOREIGN KEY. Вычисление производится согласно формуле (12).

$$maintenance\_work\_mem = 0.05 \times total\_ram\_size,$$
(12)
где total_ram_size - общий объем памяти на сервере СУБД.

Параметр autovacuum_work_mem задает объем памяти, потребляемой каждым рабочим процессом автоматической очистки. Вычисление основано на данных, полученных от приложения-агента, а также на значении параметра autovacuum_max_workers.

$$autovacuum\_work\_mem = 0.1 \times total\_ram \div autovacuum\_max\_workers,$$
(13)
где total_ram - общий объем памяти на сервере СУБД,
autovacuum_max_workers - текущее значение одноименного параметра.

Параметр max_worker_processes определяет максимальное число фоновых процессов, которое можно запустить в текущей системе. Значение по умолчанию равно восьми. В ходе вычислений данному параметру присваивается значение, равное числу логических процессоров на сервере СУБД.

$$max\_worker\_processes = number\_of\_cpu\_cores,$$
(14)
где number_of_cpu_cores - число процессорных ядер на сервере СУБД.

Параметр max_parallel_workers задаёт максимальное число рабочих процессов, которые могут выполняться на сервере СУБД одновременно. Значение по умолчанию - 8. Данному параметру присваивается значение, равное числу логических процессоров на сервере СУБД.

$$max\_parallel\_workers = number\_of\_cpu\_cores,$$
(15)

Где number_of_cpu_cores - число процессорных ядер на сервере СУБД.

Параметр max_parallel_workers_per_gather определяет максимальное число процессов, доступных СУБД для запуска операций Gather и Gather Merge (узлы плана запроса). Значение по умолчанию равно двум. Вычисление выполняется согласно формуле (16).

$$max\_parallel\_workers\_per\_gather = min(4,\ 0.5\ *\ number\_of\_cpu\_cores),$$
(16)

где number_of_cpu_cores - число процессорных ядер на сервере СУБД.

Параметр max_parallel_maintenance_workers определяет максимальное число процессов, доступных в рамках выполнения одной системной операции, такой как CREATE INDEX и VACUUM. Значение по умолчанию равно двум. В ходе процесса вычисления данному параметру присваивается значение согласно формуле (17).

$$max\_parallel\_maintenance\_workers = min(4,\ 0.5\ *\ number\_of\_cpu\_cores)$$
, (17)

где number_of_cpu_cores - число процессорных ядер на сервере СУБД.

4.5.4. Параметры, связанные с построением планов запросов

Параметр effective_cache_size определяет представление планировщика об эффективном размере дискового кэша, доступном для одного запроса.

Вычисление основано на данных, полученных от приложения-агента и выражено в формуле (18).

$$effective\_cache\_size = 0.75 \times total\_ram\_size,$$

(18)

где total_ram_size - общий объем памяти на сервере СУБД.

4.5.5. Параметры, связанные с ведением статистики

Параметр track_counts включает внутренний процесс сбора статистики, необходимой для корректной работы автоочистки. Данный параметр всегда устанавливается инструментом в true ввиду того, что инструмент выполняет вычисления в контексте автоматической очистки.

$$track\_counts = true$$

(19)

4.5.6. Параметры, связанные с ведением журнала предзаписи

Параметр checkpoint_timeout задает временной интервал между двумя соседними контрольными точками. Значение по умолчанию равно 5 мин. и является слишком низким для большинства современных конфигураций серверов СУБД. Согласно формуле (20), данному параметру инструментом всегда присваивается значение в 30 минут.

$$checkpoint\_timeout = 30min$$
(20)

Параметр max_wal_size определяет максимальный размер журнала предварительной записи, при превышении которого внутренними процессами PostgreSQL принудительно запускается фиксация контрольной точки (checkpoint).

Вычисление max_wal_size основано на получении среднего объема трафика, попадающего в журнал предзаписи за единицу времени. Информация об объеме трафика формируется исходя из данных в таблице wal_lsn_history системного хранилища данных. Процесс заполнения системной таблицы описан в разделе 4.2. Далее приведен пример вычисления среднего объема WAL-трафика.

Таблица 6. Пример выборки из таблицы wal_lsn_history

Дата и время сохранения записи	**Смещение в журнале предзаписи**
2021-06-01 09:00:00	32/A0000000
2021-06-01 09:01:00	32/A0001000
2021-06-01 09:02:00	32/A0002000

В выборке, представленной в таблице 6, фиксация текущего смещения (в байтах) в журнале предзаписи выполнялась 1 раз в минуту, что является поведением инструмента по умолчанию. При этом скорость заполнения журнала предзаписи за период между 2021-06-01 09:00:00 и 2021-06-01 09:01:00 составила 4096 байт в минуту.

Разница в байтах между двумя соседними записями в таблице wal_lsn_history является элементом выборки при расчете скользящего среднего. Размер скользящего окна конфигурируется при помощи параметра центральной службы RollingWindow (WAL). Конфигурация инструмента описана в разделе 4.4 данной работы.

$$max\_wal\_size = \frac{\sum_{i=0}^{n} x_i}{n} \times C \times (1 + T),$$

(21)

где x_i - i-ый элемент среди выборки, полученной исходя из данных в таблице wal_lsn_history согласно формуле (22),

C - текущее значение параметра checkpoint_timeout,

T - текущее значение параметра checkpoint_completion_target.

$$x_i = \frac{wal\_insert\_location_{i+1} - wal\_insert\_location_i}{log\_timestamp_{i+1} - log\_timestamp_i},$$

(22)

где $wal\_insert\_location_i$ - значение смещения в таблице wal_lsn_log по индексу i,

$log\_timestamp_i$ - дата и время фиксации значения смещения с индексом i.

Параметр checkpoint_warning определяет временной интервал, условно ограничивающий снизу расстояние во времени между запусками процесса создания контрольной точки (checkpoint). Если две контрольные точки создаются ближе во времени относительно друг друга, в журнал PostgreSQL попадает сообщение с предупреждением. Вычисление зависит от значения параметра checkpoint_timeout согласно формуле (23).

$$checkpoint\_warning = 0.8 \times checkpoint\_timeout,$$

(23)

где checkpoint_warning - текущее значение одноименного параметра.

Параметр checkpoint_completion_target определяет долю от значения параметра checkpoint_timeout для определения временного интервала, в который должен уложиться процесс создания контрольной точки. Вычисление параметра зависит от checkpoint_timeout согласно формуле (24).

$$checkpoint\_completion\_target = min(0.9, \frac{checkpoint\_timeout - 2\,min}{checkpoint\_timeout}),$$

(24)

где checkpoint_timeout - текущее значение одноименного параметра.

Параметр wal_buffers определяет объем памяти, выделяемой в разделяемом кэше на уровне оперативной памяти (shared buffers) для буферизации данных журнала предварительной записи, еще не записанных на диск. Вычисление зависит от значения параметра shared_buffers и производится согласно формуле (25).

$$wal\_buffers = min(max(shared\_buffers / 32,\ 64kB),\ 32MB),$$

(25)

где shared_buffers - значение одноименного параметра

4.6. Графический интерфейс

Данный раздел посвящен описанию графического интерфейса, предоставляемого приложением управления. Интерфейс реализован с использованием сторонней библиотеки Terminal.Gui и состоит из элементов, отображаемых в окне командной строки. Выбор такого способа взаимодействия с пользователем обоснован тем, что инструмент должен быть расположен в одном сетевом контуре с конфигурируемыми серверами СУБД, что накладывает ограничения на доступ к API инструмента извне. Отображение же интерфейса в командной строке позволяет управлять инструментом и получать необходимые данные посредством подключения к серверу по протоколу SSH (Secure Shell).

Ниже на рисунке 7 приведен снимок экрана, демонстрирующий главное меню приложения управления.

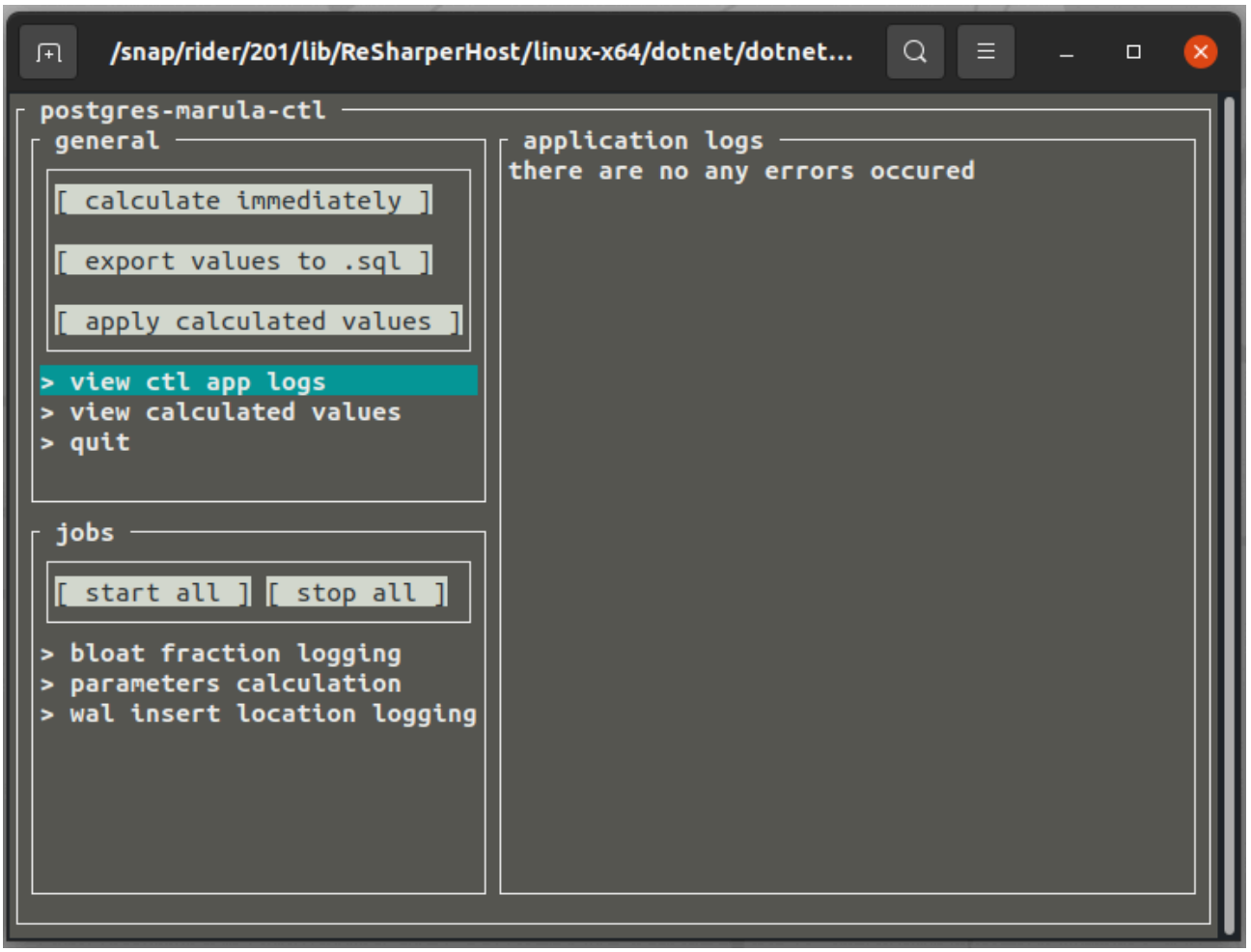

Рисунок 7. Главное меню приложения управления

Окно графического интерфейса разделено на три области:

- Основные элементы управления (general).
- Элементы управления периодическими процессами центральной службы (jobs).
- Правая часть окна, на которой отображается информация в зависимости от контекста.

Основные элементы управления предоставляют следующие возможности:

- Принудительный запуск вычисления параметров независимо от периодического процесса.

- Формирования SQL-скрипта, содержащего набор команд ALTER SYSTEM для каждого вычисленного значения параметра (рисунок 8).
- Применение наиболее актуального набора значений параметров к конфигурации сервера СУБД.
- Просмотр журнала информационных сообщений и сообщений об ошибках приложения управления.
- Просмотр актуальных значений параметров (вычисленных в ходе последней итерации периодического процесса, либо вычисленных принудительно по нажатию кнопки «calculate immediately». Реализация чтения значений из системного хранилища приведена в приложении В. Значения отображаются в правой части окна. Пример отображения приведен на рисунке 9.

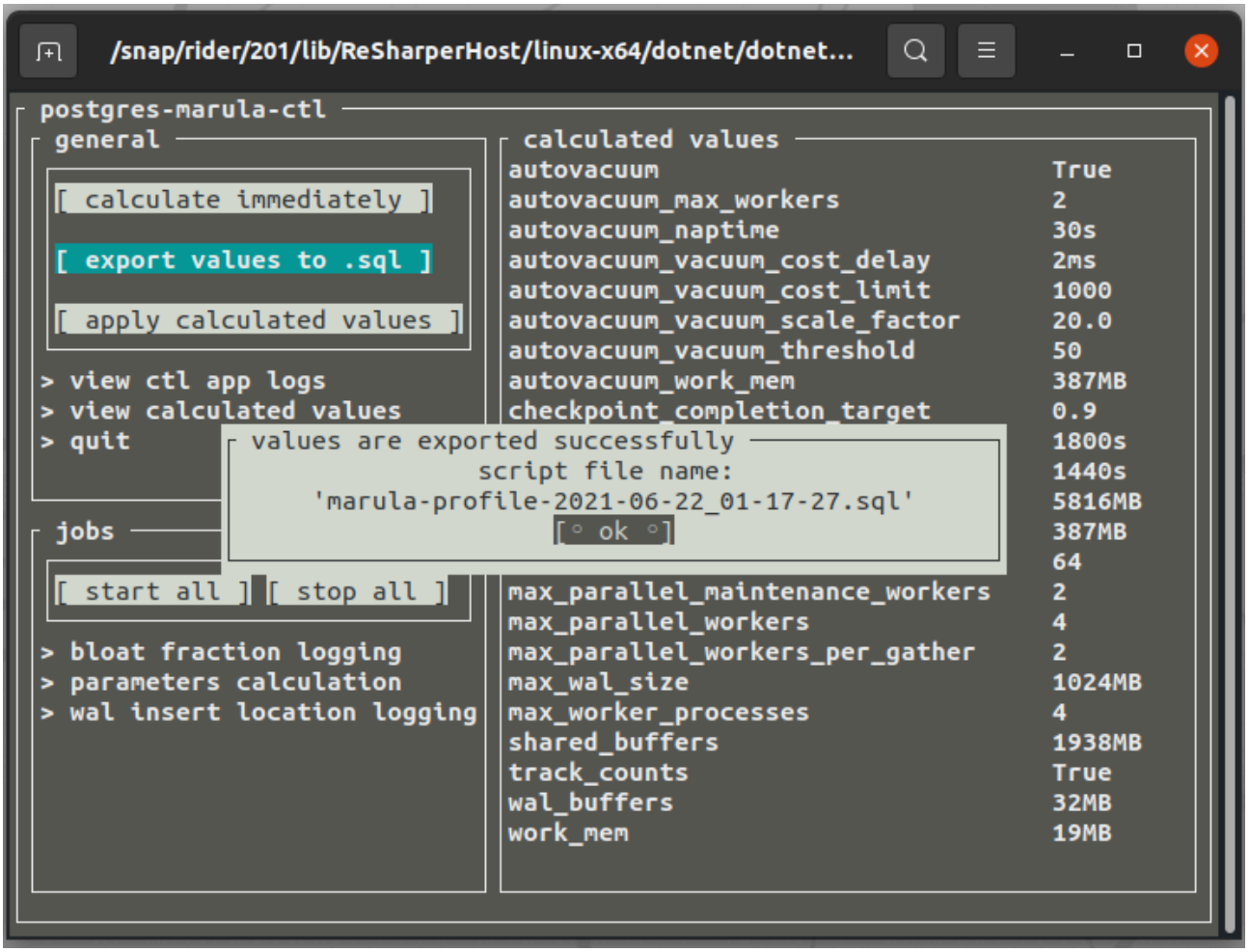

Рисунок 8. Экспорт значений параметров в файл с расширением .sql

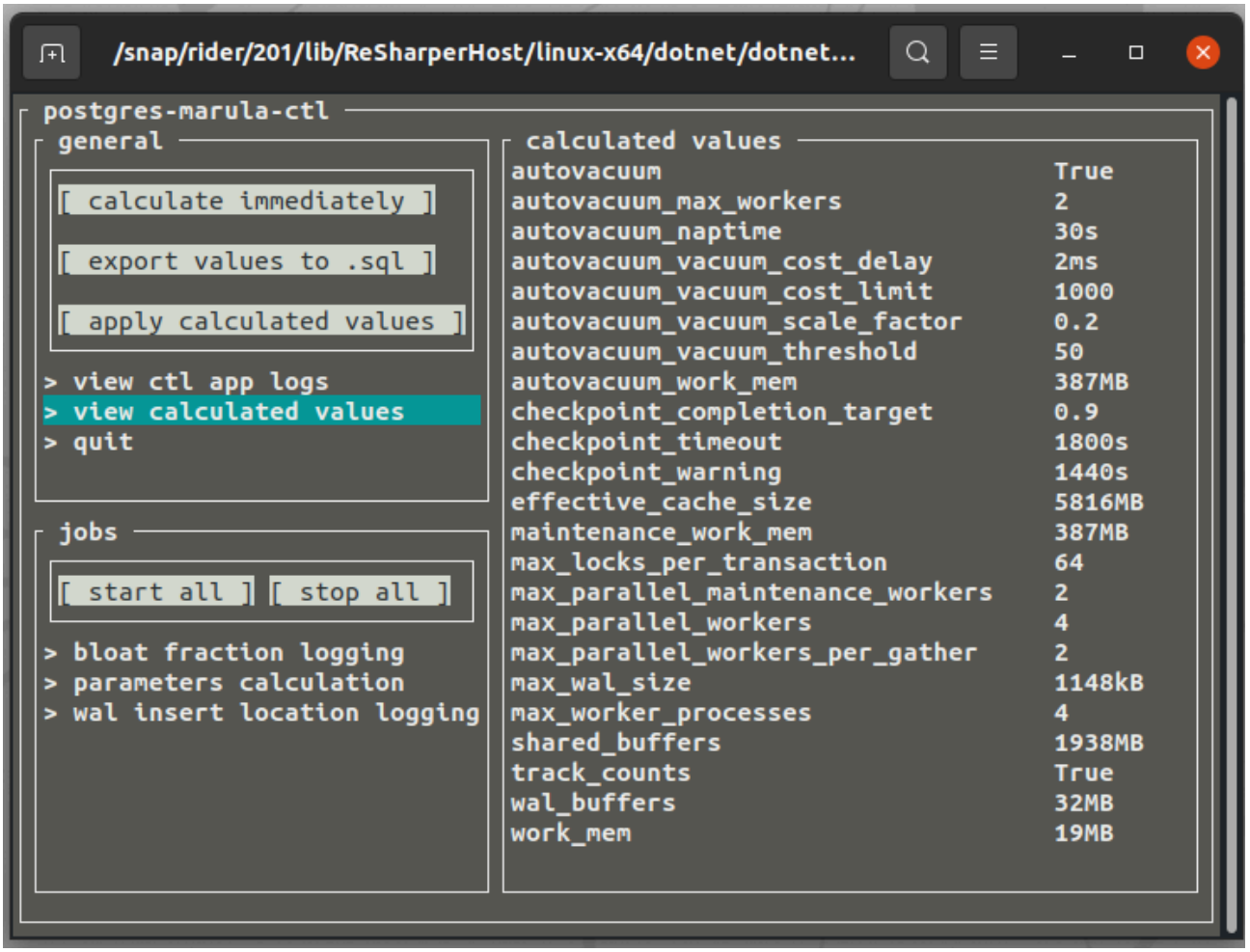

Рисунок 9. Отображение вычисленных значений в графическом интерфейсе приложения управления

Элементы управления периодическими процессами позволяют запускать и останавливать процессы вычисления параметров, логирования текущего смещения в журнале предзаписи и логирования среднего числа записей, подлежащих удалению. На рисунке 10 представлен пример запуска.

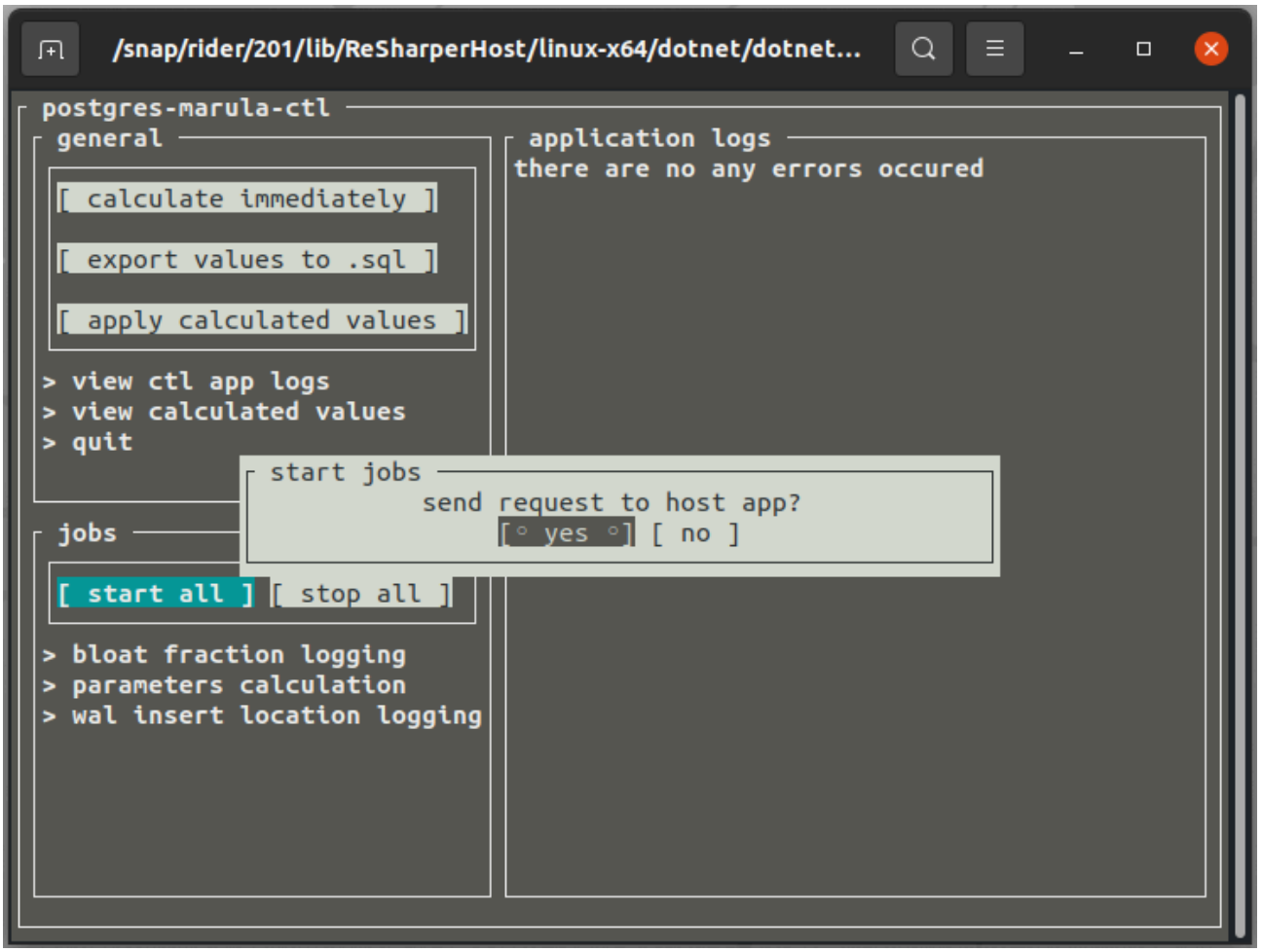

Рисунок 10. Запуск периодических процессов через графический интерфейс приложения управления

В данном случае по нажатию кнопки «yes» приложением управления будет отправлен HTTP-запрос центральной службе, инициирующий запуск всех периодических процессов.

ГЛАВА 5. ТЕСТИРОВАНИЕ ПРОИЗВОДИТЕЛЬНОСТИ СЕРВЕРА СУБД

В данной главе приведено сравнение производительности сервера СУБД при разных наборах конфигурационных параметров. Производительность исчисляется в количестве выполняемых сервером транзакций в секунду (TPS), а также в среднем времени отклика сервера на поступающие от приложений-клиентов запросы (latency).

5.1. Описание тестового окружения и сценария тестирования

Тестирование производилось при помощи стандартного инструмента Pgbench, входящего в комплект поставки PostgreSQL.

Тестовое окружение:

- Операционная система Ubuntu 20.04
- Сервер СУБД PostgreSQL 13.3
- Виртуальный сервер AWS t2.xlarge (4 CPU, 16GB RAM)

Таблица 7. Параметры инициализации тестовой схемы данных (pgbench --initialize)

Имя параметра	Значение
scale	1000

Таблица 8. Параметры запуска теста производительности (pgbench)

Имя параметра	Значение
client	50
jobs	4
time	3600
progress	360

В таблицах 7 и 8 указаны параметры запуска утилиты pgbench для инициализации тестовой схемы данных и запуска теста производительности.

5.2. Сравнение производительности СУБД при различных профилях конфигурации

Приведено сравнение следующих профилей конфигурации:

1. Конфигурация сервера по умолчанию.
2. Конфигурация, полученная в результате работы приложения Pgtune, описанного в разделе 2.1 данной работы.
3. Конфигурация, полученная в результате анализа сервера разработанным инструментом.

На рисунке 11 изображена диаграмма, иллюстрирующая данные по TPS (transactions per second), полученные в результате выполнения тестового сценария при трех различных профилях конфигурации. Для изоляции итераций друг от друга после каждого завершения выполнения сценария был выполнен перезапуск сервера.

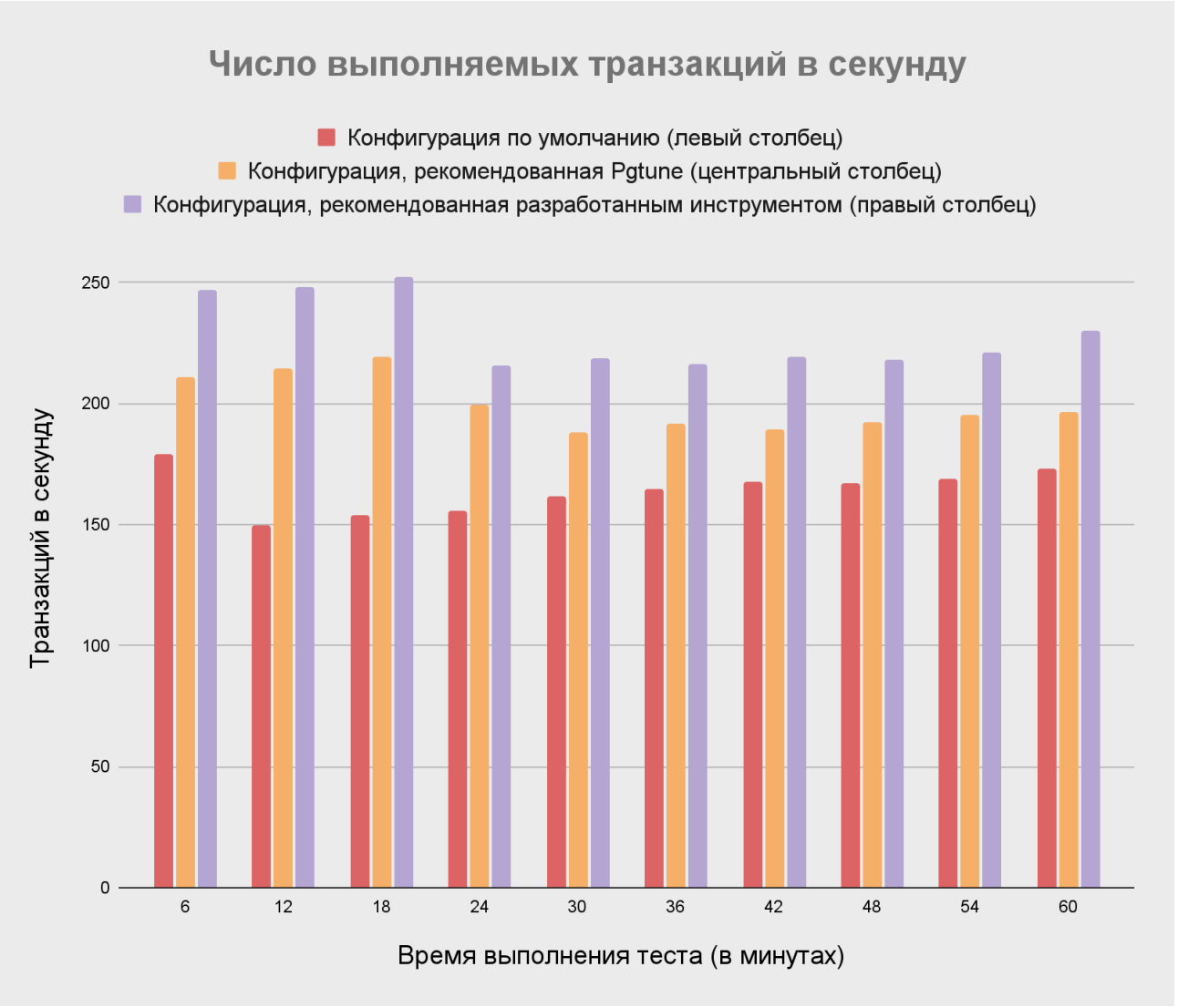

Рисунок 11. Сравнение числа транзакций в секунду, выполняемых сервером СУБД

Среднее значение TPS при конфигурации по умолчанию составило 164,22, при конфигурации, рекомендованной Pgtune - 199,87, при конфигурации, рекомендованной разработанным инструментом - 228,83.

На рисунке 12 представлена диаграмма, аналогичная диаграмме на рисунке 11, но демонстрирующая сравнение среднего времени отклика в миллисекундах.

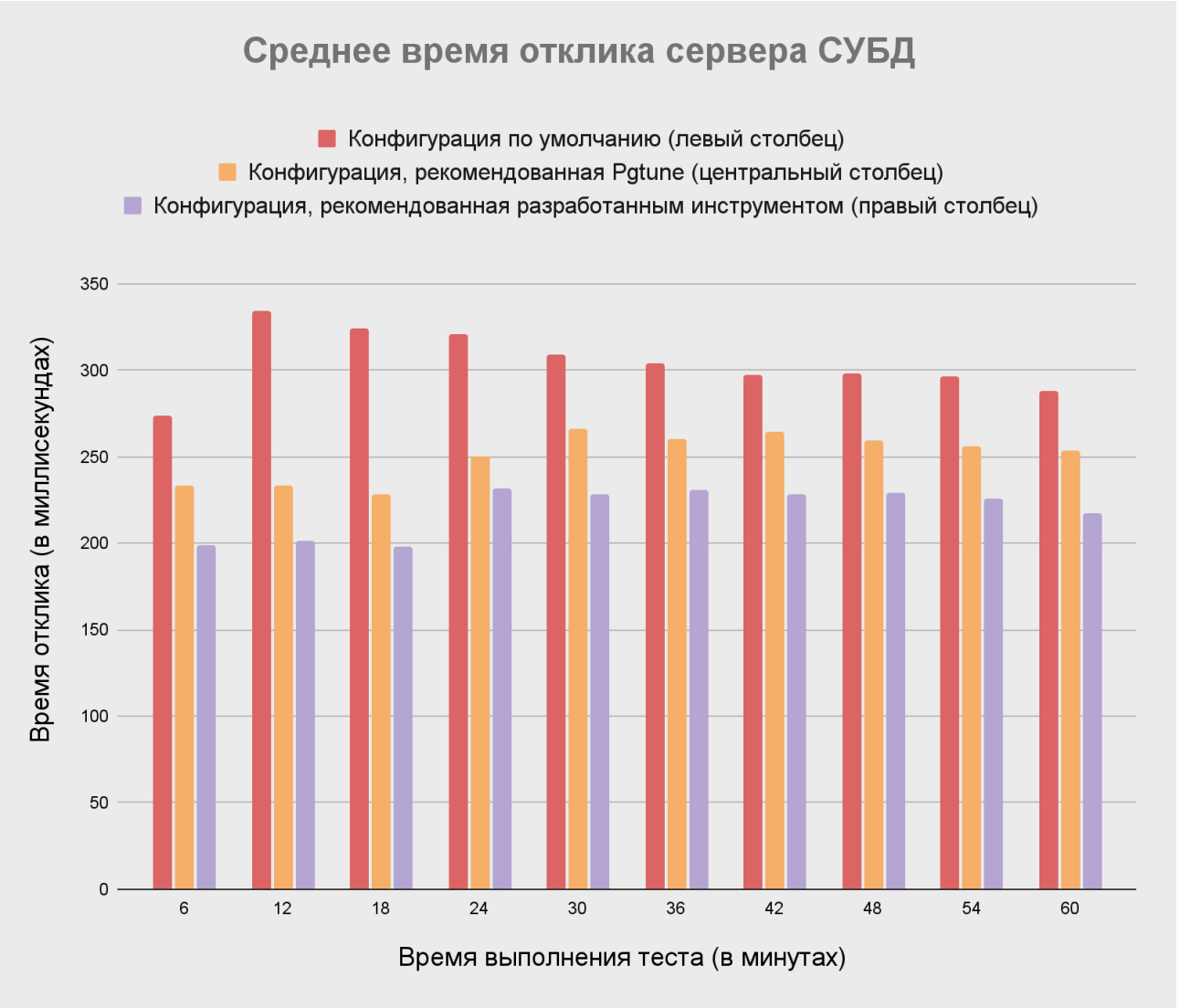

Рисунок 12. Сравнение среднего времени отклика СУБД на входящие запросы

Среднее время отклика при конфигурации по умолчанию составило 304,83 мс, при конфигурации, рекомендованной Pgtune - 250,44 мс, при конфигурации, рекомендованной разработанным инструментом - 218,96 мс.

По полученным в ходе тестирования данным можно сделать вывод, что анализ статистических данных СУБД помогает выполнять более качественные вычисления параметров, что в свою очередь положительно сказывается на производительности СУБД.

Прирост производительности по сравнению с конфигурацией, рекомендованной инструментом Pgtune, составил около 15%, а по сравнению с конфигурацией по умолчанию - около 40%.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы был разработан инструмент для динамической и рекомендательной конфигурации сервера СУБД PostgreSQL.

Выполнены все поставленные задачи:

- Организация доступа к статистическим данным сервера PostgreSQL.
- Вычисление значений параметров в конкретный момент времени на основе полученных данных.
- Применение полученных значений к конфигурации сервера СУБД как в автоматическом, так и в ручном режиме.
- Автоматические повторные вычисления, выполняемые с задаваемым интервалом.
- Сбор дополнительной необходимой статистики, не предоставляемой сервером СУБД по умолчанию.

Сравнительное тестирование, описанное в главе 5, показало, что прирост производительности при применении разработанного инструмента может составлять от 10% до 40% как по сравнению с профилем конфигурации по умолчанию, так и при использовании параметров, рекомендованных инструментом Pgtune.

Программный код разработанного инструмента находится в открытом доступе на платформе Github: https://github.com/zadykian/postgres-marula.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Генетический оптимизатор запросов // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/geqo (дата обращения: 10.02.2021).
2. Журнал предзаписи // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/wal-intro (дата обращения: 16.01.2021).
3. Логическая репликация // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/logical-replication (дата обращения: 12.03.2021).
4. Новиков Б. А. Основы Технологий Баз Данных. - ДМК, 2020. – 620 с.
5. Примеры оценки количества строк // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/row-estimation-examples (дата обращения: 18.02.2021).
6. Работа с PostgreSQL. Настройка и масштабирование. Васильев А. Ю. [Электронный ресурс]. - URL: https://postgresql.leopard.in.ua/ (дата обращения: 11.03.2021).
7. Системные каталоги // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/catalogs (дата обращения: 14.02.2021).
8. Физическое хранение базы данных // PostgresPro [Электронный ресурс]. - URL: https://postgrespro.ru/docs/postgresql/13/storage (дата обращения: 11.02.2021).
9. Appigatla K. MySQL 8 Cookbook: Over 150 recipes for high-performance database querying and administration. - Packt Publishing Ltd, 2018. – 560 p.
10. B-Tree Indexes Implementation // PostgreSQL [Электронный ресурс]. - URL: https://www.postgresql.org/docs/13/btree-implementation.html (дата обращения: 25.03.2021).

11. Database Performance Tuning Guide // Oracle Database [Электронный ресурс]. – URL: https://docs.oracle.com/en/database/oracle/oracle-database/21/tgdba/index.html (дата обращения: 25.12.2020).

12. Garcia-Molina H. Database Systems: The Complete Book. - Pearson Prentice Hall, 2009. - 1248 p.

13. Shaik B. PostgreSQL Configuration: Best Practices for Performance and Security. - Apress, 2020. - 312 p

14. Vanier E., Shah B., Malepati T. Advanced MySQL 8: Discover the full potential of MySQL and ensure high performance of your database. - Packt Publishing Ltd, 2019. - 286 p.

15. Wisborg Krogh J. MySQL 8 Query Performance Tuning: A Systematic Method for Improving Execution Speeds. - Apress, 2020. - 965 p.

ПРИЛОЖЕНИЕ А

```
/// <summary>
/// Получение среднего числа записей, подлежащих удалению.
/// <summary/>
async Task<Fraction>
    IDatabaseServer.GetAverageBloatFractionAsync()
{
    var queryText = string.Intern(@"
        select avg(n_dead_tup / (n_dead_tup + n_live_tup))
        from pg_catalog.pg_stat_all_tables
        where n_live_tup + n_dead_tup != 0;");

    var connection = await Connection();
    return await connection
        .ExecuteScalarAsync<Fraction>(queryText);
}
```

ПРИЛОЖЕНИЕ Б

```
/// <summary>
/// Получение иерархических связей между таблицами.
/// <summary/>
async IAsyncEnumerable<ParentToChild>
    IDatabaseServer.GetAllHierarchicalLinks()
{
    var queryText = string.Intern($@"
        select
            concat_ws('.',
                parent_class.relnamespace::regnamespace,
                parent_class.relname)
                as {nameof(ParentToChild.Parent)},
            concat_ws('.',
                child_class.relnamespace::regnamespace,
                child_class.relname)
                as {nameof(ParentToChild.Child)}
        from pg_catalog.pg_inherits
        inner join pg_catalog.pg_class parent_class
            on pg_inherits.inhparent = parent_class.oid
        inner join pg_catalog.pg_class child_class
            on pg_inherits.inhrelid = child_class.oid
        where parent_class.relkind in ('p', 'r');");

    var connection = await Connection();
    var parentToChildLinks = await connection
        .QueryAsync<ParentToChild>(queryText);
    foreach (var parentToChild in parentToChildLinks)
        yield return parentToChild;
}
```

ПРИЛОЖЕНИЕ В

```
/// <summary>
/// Получение наиболее актуальных значений параметров
/// из системного хранилища данных.
/// <summary/>
async IAsyncEnumerable<IValueView>
    IParameterValues.MostRecentAsync()
{
    var queryText = string.Intern($@"
        with ranked_values as
        (
            select
                parameter_id,
                calculated_value,
                row_number() over (
                    partition by parameter_id
                    order by calculation_timestamp desc) as rank
            from marula_tool.parameters_values_history as history
        )
        select
            parameters.name
                as {nameof(IValueView.Link)},
            ranked_values.calculated_value
                as {nameof(IValueView.Value)}
        from ranked_values
        inner join marula_tool.calculated_parameters as parameters
            on ranked_values.parameter_id = parameters.id
        where ranked_values.rank = 1;");

    var connection = await Connection();
    var parameterValues = await connection
        .QueryAsync<(IParameterLink, NonEmptyString)>(queryText);

    foreach (var (link, value) in parameterValues)
        yield return await valueViewFactory
            .CreateAsync(link, value);
}
```