Russian Technological Journal.
2024;12(3):7–24

ISSN 2782-3210 (Print)
ISSN 2500-316X (Online)

Информационные системы. Информатика. Проблемы информационной безопасности

Information systems. Computer sciences. Issues of information security

УДК 004.657
https://doi.org/10.32362/2500-316X-2024-12-3-7-24
EDN BNQNDI

НАУЧНАЯ СТАТЬЯ

Оценка эффективности балансировщика соединений *PgBouncer* для оптимизации вычислительных ресурсов реляционных баз данных

А.С. Боронников [@],
П.С. Цынгалёв,
В.Г. Ильин,
Т.А. Деменкова

МИРЭА – Российский технологический университет, Москва, 119454 Россия
[@] Автор для переписки, e-mail: boronnikov-anton@mail.ru

Резюме

Цели. Целью работы является исследование возможностей использования балансировщика подключений *PgBouncer* с различными конфигурациями в современных инсталляциях баз данных (БД) путем проведения нагрузочного тестирования с различными сценариями, максимально приближенными к реальной нагрузке, определение критичных показателей, получение результатов тестирования и интерпретация их в виде графиков.

Методы. В ходе исследования использовались методы эксперимента, индукции, тестирования и статистического анализа.

Результаты. Рассмотрены основные возможности, архитектура и режимы работы сервиса *PgBouncer*. Проведено нагрузочное тестирование на виртуальной машине, развернутой на базе открытой облачной платформы, с различной конфигурацией затрачиваемых вычислительных ресурсов – центрального процессора (CPU), оперативной памяти (RAM) и использованием нескольких сценариев с разной конфигурацией и разным количеством подключений балансировщика к БД. В ходе тестирования были исследованы основные показатели: распределение использования процессора, утилизация оперативной памяти, дискового пространства и центрального процессора. Выполнены интерпретация полученных данных и анализ полученных результатов путем выделения критических параметров. Сформулированы выводы и рекомендации по использованию балансировщика подключения в реальных высоконагруженных инсталляциях для оптимизации утилизируемых ресурсов сервером, на котором расположена система управления базами данных (СУБД). Сформировано заключение о полезности использования балансировщика запросов *PgBouncer* и предложены варианты конфигурации для последующего использования в реальных инсталляциях.

Выводы. Исследована степень влияния использования балансировщика соединений *PgBouncer* на производительность системы в целом, развернутой в виртуализированной среде. Результаты работы показали, что применение *PgBouncer* позволяет существенно оптимизировать затрачиваемые вычислительные ресурсы вычислительного узла под сервер СУБД, а именно: уменьшилась нагрузка на CPU на 15%, на RAM – на 25–50%, на дисковую подсистему – на 20%, в зависимости от сценариев тестов, количества подключений к БД, конфигурации балансировщика подключений.

Ключевые слова: PgBouncer, PostgreSQL, пуллер, балансировщик, база данных, оптимизация, мониторинг, виртуальная машина, облачные технологии

• **Поступила:** 13.06.2023 • **Доработана:** 06.12.2023 • **Принята к опубликованию:** 09.04.2024

Для цитирования: Боронников А.С., Цынгалёв П.С., Ильин В.Г., Деменкова Т.А. Оценка эффективности балансировщика соединений *PgBouncer* для оптимизации вычислительных ресурсов реляционных баз данных. *Russ. Technol. J*. 2024;12(3):7–24. https://doi.org/10.32362/2500-316X-2024-12-3-7-24

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

RESEARCH ARTICLE

Evaluation of connection pool *PgBouncer* efficiency for optimizing relational database computing resources

Anton S. Boronnikov [@], **Pavel S. Tsyngalev**, **Victor G. Ilyin**, **Tatiana A. Demenkova**

MIREA – Russian Technological University, Moscow, 119454 Russia
[@] *Corresponding author, e-mail: boronnikov-anton@mail.ru*

Abstract

Objectives. The aim of the research is to investigate the possibilities of using the *PgBouncer* connection pool with various configurations in modern database installations by conducting load testing with diverse real-world like scenarios, identifying critical metrics, obtaining testing results, and interpreting them in the form of graphs.

Methods. The research utilized methods of experimentation, induction, testing, and statistical analysis.

Results. The main features, architecture and modes of operation of the *PgBouncer* service are considered. Load testing was carried out on a virtual machine deployed on the basis of an open cloud platform with different configurations of computing resources (CPU, RAM) and according to several scenarios with different configurations and different numbers of balancer connections to the database, during which the following main indicators were investigated: distribution of processor usage, utilization of RAM, disk space, and CPU. The interpretation of the data obtained and the analysis of the results obtained by highlighting critical parameters are performed. On the basis of results analysis, conclusions and recommendations are formulated on the use of a connection balancer in real high-load installations for optimizing the resources utilized by the server on which the database management system (DBMS) is located. A conclusion is presented on the usefulness of using the *PgBouncer* query balancer along with proposed configuration options for subsequent use in real installations.

Conclusions. The degree of influence of the use of the *PgBouncer* connection balancer on the performance of the system as a whole deployed in a virtualized environment is investigated. The results of the work showed that the use of *PgBouncer* allows significantly optimization of the computing resources of a computing node for a DBMS server, namely, load on the CPU decreased by 15%, RAM—by 25–50%, disk subsystem—by 20%, depending on the test scenarios, the number of connections to the database, and the configuration of the connection balancer.

Keywords: PgBouncer, PostgreSQL, connection pool, balancer, databases, optimization, monitoring, virtual machines, cloud technologies

• **Submitted:** 13.06.2023 • **Revised:** 06.12.2023 • **Accepted:** 09.04.2024

For citation: Boronnikov A.S., Tsyngalev P.S., Ilyin V.G., Demenkova T.A. Evaluation of connection pool *PgBouncer* efficiency for optimizing relational database computing resources. *Russ. Technol. J*. 2024;12(3):7–24. https://doi.org/10.32362/2500-316X-2024-12-3-7-24

Financial disclosure: The authors have no a financial or property interest in any material or method mentioned.

The authors declare no conflicts of interest.

ВВЕДЕНИЕ

Базы данных (БД) являются неотъемлемой частью современных приложений. Они используются для хранения, управления и обработки большого объема информации. Одной из главных проблем, с которыми сталкиваются приложения, является управление множественными подключениями к БД.

Подключение к БД – это процесс установления связи между клиентским приложением и сервером БД. Каждое клиентское приложение устанавливает свое подключение к БД. Это может привести к излишней нагрузке на сервер БД и снижению производительности приложения. Кроме того, каждое подключение к БД требует определенных ресурсов, таких как память и процессорное время. Если значительное количество клиентских приложений одновременно устанавливает подключение к БД, то это может привести к перегрузке сервера и снижению производительности приложения.

Для решения проблемы можно оптимизировать саму систему управления базами данных (СУБД) путем конфигурирования ее параметров на этапе запуска инфраструктуры [1–3] или же использовать сторонние сервисы – балансировщики подключений в БД. Они позволяют управлять подключениями клиентов, чтобы максимально использовать ресурсы сервера БД, тем самым повышая производительность приложения. Существует несколько типов балансировщиков [4]. В данной статье рассматриваются основные возможности инструмента, который относится к типу балансировки подключений к БД на прикладном уровне (application-level balancing) – *connection pool* (набор, пул соединений). В русскоязычной литературе не имеется точного понятия для обозначения таких агрегаторов запросов к БД, поэтому в рамках данной публикации введен термин «*пуллер соединений*» или просто «*пуллер*».

1. ПУТЬ ЗАПРОСОВ К БД

Внедрение пуллера приводит к значительным изменениям в работе с БД. Чтобы их заметить, необходимо предварительно изучить стандартный маршрут прохождения запросов. В обычной архитектуре клиент-серверных соединений (client-server) имеет место следующий стандартный принцип взаимодействия, изображенный на рис. 1.

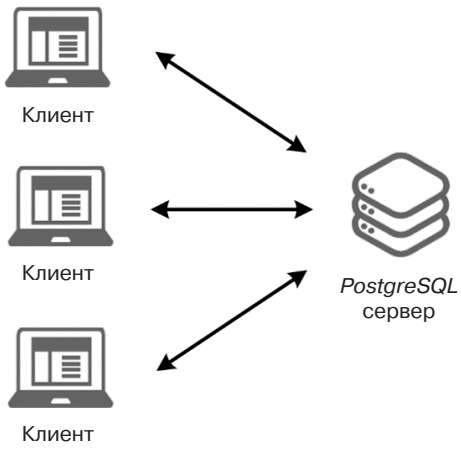

Рис. 1. Обычное клиент-серверное соединение[1]

При установлении новой сессии клиентское приложение запрашивает соединение с сервером, и проходит процесс аутентификации. Сервер в ответе создает отдельный системный процесс для обработки соединения и работы сессии. Инициализация состояния сессии осуществляется на основе различных параметров конфигурации, определенных на уровне сервера, БД и пользователя. В рамках одной сессии клиент выполняет требуемые операции. Работа продолжается до тех пор, пока клиент не завершит сессию путем отключения. После завершения сессии сервер уничтожает соответствующий системный процесс, ответственный за обработку данной сессии.

Можно выделить недостатки обычного клиент-серверного соединения:

1) создание, управление и удаление процессов соединения занимает время и расходует ресурсы;
2) при увеличении числа соединений на сервере возрастает и потребность в ресурсах для их управления. Кроме того, использование памяти на сервере растет с выполнением операций клиентами;
3) поскольку одна сессия обслуживает только одного клиента, клиенты могут изменять состояние сессии БД и ожидать, что эти изменения сохранятся в последующих транзакциях.

[1] *PostgreSQL*. https://www.postgresql.org/. Дата обращения 18.04.2023. / Accessed April 18, 2023.

При использовании пуллера клиенты подключаются к нему, а он уже устанавливает соединение с сервером (рис. 2). Это изменяет модель стандартного принципа соединений на клиент-прокси-серверную архитектуру (client-proxy-server).

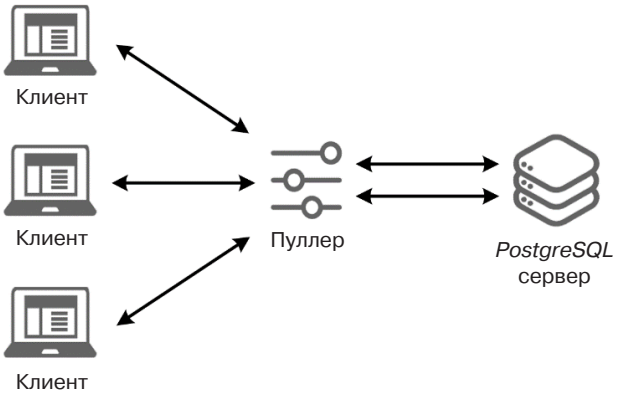

Рис. 2. Соединение «клиент-прокси-сервер»

Теперь подключение клиента к серверу не зависит от времени жизни соединения и процесса на сервере. Пуллер отвечает за принятие и управление соединениями от клиента, установление и поддержание соединений с сервером, а также за назначение серверных соединений клиентским соединениям.

Отличным выбором по качеству системы управления БД является *PostgreSQL* [5–9]. Одним из главных факторов является статус *PostgreSQL* как открытого программного обеспечения (open source). Существует несколько достойных пуллеров соединений с *PostgreSQL* [6, 10–12], таких как *PgBouncer*[2], *Pgpool-II*[3] и *Odyssey*[4]. В данной работе рассмотрены основные возможности, архитектура и режимы работы *PgBouncer*.

2. ПУЛЛЕР СОЕДИНЕНИЙ *PgBouncer*

PgBouncer – это пуллер, позволяющий управлять соединениями с БД *PostgreSQL*. Он работает как прокси-сервер, который обрабатывает запросы на подключение к БД и перенаправляет их на соответствующий сервер. *PgBouncer* может быть установлен на той же машине, что и *PostgreSQL*, либо на отдельной.

Данный пуллер широко используется во многих приложениях на базе *PostgreSQL* и применяется для решения различных задач, связанных с производительностью, масштабируемостью и безопасностью. Данный балансировщик активно используется в продуктах таких крупных компаний, как Alibaba[5], Huawei[6], Instagram[7] (запрещена в Российской Федерации), Skype[8], в т.ч. на российском рынке[9] – Яндекс[10], Avito[11], Сбербанк[12], Газпромнефть[13] и др.

Одной из главных задач *PgBouncer* является управление соединениями. Он позволяет создавать пулы соединений, которые могут быть использованы несколькими клиентами. Это позволяет снизить нагрузку на сервер БД и повысить производительность приложения.

2.1. Архитектура

В официальной документации *PgBouncer* отсутствует описание архитектуры балансировщика. Был проведен анализ библиотек данного пуллера соединений и исследована его функциональность. На основе собственного реверс-инжиниринга предложена архитектура *PgBouncer*, изображенная на рис. 3.

Слушатель играет важную роль в обработке соединений клиентов с БД *PostgreSQL*. Он обеспечивает точку входа для клиентских подключений (так называемый *сокет*) и выполняет роль посредника между клиентом и сервером. Слушатель включает в себя еще *протокол,* который определяет формат обмена данными между клиентом и сервером через сокет. *PgBouncer* использует тот же протокол, что и *PostgreSQL*, но, помимо этого, имеет собственные расширения и дополнительные команды.

Аутентификация обеспечивает проверку подлинности клиента при попытке подключения к нему. Поддерживаются различные методы, такие как md5, trust, plain, cert и др.

---

[2] Официальная документация *PgBouncer* [Official documentation *PgBouncer*]. https://www.pgbouncer.org/. Дата обращения 02.04.2023. / Accessed April 02, 2023.

[3] *Pgpool* Wiki. https://pgpool.net/mediawiki/index.php/Main_Page. Дата обращения 15.04.2023. / Accessed April 15, 2023.

[4] *Odyssey* – Yandex Technologies. https://yandex.ru/dev/odyssey/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

[5] https://www.alibaba.com/. Дата обращения 15.04.2023. / Accessed April 15, 2023.

[6] https://www.huawei.com/. Дата обращения 15.04.2023. / Accessed April 15, 2023.

[7] https://www.instagram.com/. Дата обращения 15.04.2023. / Accessed April 15, 2023.

[8] https://www.skype.com/ru/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

[9] Почему крупнейшие компании России и мира выбирают Postgres. Итоги PgConf.Russia 2017. http://www.interface.ru/home.asp?artId=39028. Дата обращения 10.04.2023. [Why the largest companies in Russia and the world choose Postgres. Results of PgConf.Russia 2017. http://www.interface.ru/home.asp?artId=39028 (in Russ.). Accessed April 10, 2023.]

[10] https://yandex.ru/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

[11] https://www.avito.ru/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

[12] http://www.sberbank.ru/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

[13] https://www.gazprom-neft.ru/ (in Russ.). Дата обращения 15.04.2023. / Accessed April 15, 2023.

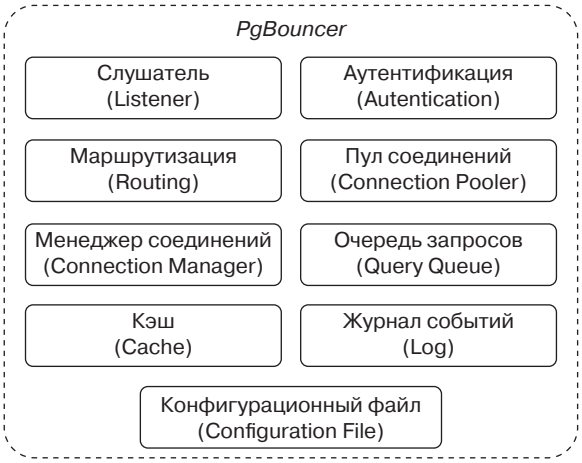

Рис. 3. Архитектура *PgBouncer*

Маршрутизация определяет режимы работы пуллера. Всего выделяют четыре типа: пул сеансов (session pooling), пул транзакций (transaction pooling), пул операторов (statement pooling) и комбинированный метод (combined pooling). Они подробнее рассматриваются в разделе 2.2.

Пул соединений представляет собой набор доступных соединений, который может быть использован клиентом для выполнения операций в БД. Если соединение свободно, то клиент может взять его из пула. Если же соединения нет, пуллер может создать новое.

Менеджер соединений отвечает за управление жизненным циклом каждого соединения в пуле. Он отслеживает состояние каждого активного соединения к БД, включая его открытие, закрытие и переиспользование свободных соединений.

Очередь запросов управляет запросами, поступающими от клиентов, когда все соединения заняты. Это позволяет обеспечить справедливую обработку запросов и избежать блокировок или перегрузки БД. Очередь запросов имеет настраиваемые параметры, которые позволяют контролировать количество и время ожидания запросов в очереди.

Кэш представляет собой дополнительный компонент, который хранит результаты предыдущих запросов, чтобы при повторном выполнении запроса не обращаться к БД, а возвращать результат непосредственно из кэша. Кэш позволяет сэкономить время и ресурсы на выполнение запросов, особенно если они являются часто повторяющимися и их результаты не изменяются.

Журнал событий ведет запись событий, таких как установление и разрыв соединений, выполнение запросов и других операций и т.д. Он позволяет анализировать и отслеживать работу пуллера, обнаруживать различные ошибки и предупреждения, мониторить производительность и проводить отладку.

Далее следует непосредственно *конфигурационный файл*, в котором можно настроить и указать параметры всех компонентов архитектуры, рассмотренных в этом разделе.

При правильной настройке этих элементов обеспечивается эффективное использование ресурсов сервера, повышается производительность и улучшается работа приложений, использующих *PostgreSQL*.

2.2. Режимы работы

Режимы работы определяют, каким образом пуллер будет управлять соединениями. В разных режимах *PgBouncer* может оказывать различное влияние на производительность и функциональность системы.

Режим сеансов (session mode) – стандартный подход, который является наиболее корректным. Заключается в том, что каждому клиенту назначается одно серверное подключение на протяжении всего времени, пока клиент остается подключенным. При отключении клиента данное подключение к серверу возвращается обратно в пул. Этот метод работы используется по умолчанию. Режим может быть полезен для приложений, имеющих много клиентских запросов, которые не являются частыми, но выполняются в рамках долгих сессий.

Режим транзакций (transaction mode) – клиенту назначается подключение к серверу только на время выполнения транзакции. При обнаружении завершения транзакции *PgBouncer* возвращает данное подключение обратно в пул. Режим может быть полезен в приложениях, которые имеют много коротких транзакций.

Режим операторов (statement mode) – самый агрессивный подход, который предполагает, что подключение к серверу будет возвращаться в пул сразу после завершения каждого запроса. В этом режиме транзакции с несколькими операторами запрещены, поскольку они не будут работать. Данный режим может быть полезен для приложений, которые имеют много повторяющихся запросов или используют запросы с одинаковой структурой.

Комбинированный режим (combined mode) – этот подход объединяет режимы транзакций и операторов. Для запросов, которые не начинают новую транзакцию, *PgBouncer* будет использовать режим операторов, а при запросах, которые начинают новую транзакцию – режим транзакций. Этот метод может быть полезен для приложений, которые выполняют множество повторяющихся запросов и требуют выполнения транзакций. Кроме того, он может быть эффективен для приложений, которые имеют большое количество уникальных запросов, но в которых транзакции могут повторяться.

Проведено сравнение режимов работы *PgBouncer* по следующим критериям: изоляция транзакций, пул соединений и производительность. Результаты исследования отображены в табл. 1.

Таблица 1. Сравнение режимов работы *PgBouncer*

Режимы работы *PgBouncer*	Изоляция транзакций	Пул соединений	Производительность
Режим сеансов	Полная	Назначается на время сеанса	Снижение из-за создания и удаления подключений
Режим транзакций	Полная	Назначается на время транзакции	Снижение из-за создания и удаления подключений
Режим операторов	Частичная	Назначается на время запроса	Повышение благодаря переиспользованию подключений
Комбинированный режим	Баланс	В зависимости от типа запроса	Баланс между производительностью и изоляцией

2.3. Особенности использования

Необходимо сделать акцент на некоторых особенностях *PgBouncer*, которые впоследствии помогут избежать ошибок при работе с ним.

Первой особенностью является ограничение по типам запросов. Некоторые запросы, такие как создание или удаление БД, не могут быть маршрутизированы через пуллер и должны быть выполнены напрямую на сервере *PostgreSQL*. Также стоит учитывать конфигурацию самой СУБД, поскольку часть параметров, например, связанных с кэшированием, могут повлиять на производительность самого пуллера. В таком случае потребуется дополнительная настройка сервера *PostgreSQL* для оптимальной работы в связке с пуллером.

Для избежания перегрузок пуллера необходимо контролировать размер пула соединений балансировщика, поскольку это может привести к переполнению и исчерпанию ресурсов системы – утилизации центрального процессора (ЦП) и оперативного запоминающего устройства (ОЗУ). Переполненный пул может привести к снижению производительности или сбоям при работе с приложениями.

При настройке расширений необходимо учитывать, что некоторые из них могут быть несовместимы с *PgBouncer*, т.к. они могут создавать свои собственные соединения к серверу *PostgreSQL*, что будет вредить производительности системы в целом. Также при использовании данного балансировщика необходимо убедиться, что он совместим с другими инструментами и технологиями, используемыми в приложении, например, с используемым ORM-фреймворком[14].

Не стоит забывать и о необходимости поддержки версии у СУБД при использовании с *PgBouncer*, поскольку некоторые версии пуллера могут не поддерживать последние версии *PostgreSQL*.

[14] Object relation mapping – технология программирования, связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая виртуальную объектную базу данных. [Object relation mapping is a programming technology that connects DBs with the concepts of object-oriented programming languages, creating a virtual object DB.]

3. ТЕСТИРОВАНИЕ И ОЦЕНКА ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Тестирование проводилось на двух виртуальных машинах (ВМ) со следующими характеристиками:

- ВМ с СУБД: 8 виртуальных центральных процессоров (vCPU)[15], 16 ГБ оперативной памяти (RAM)[16], внешний накопитель типа SSD 32 ГБ (под систему), 500 Мб/с, 320 IOPS[17], 120 ГБ (под БД), 500 Мб/с, 1200 IOPS, операционная система (ОС) Ubuntu 22.04;
- ВМ с пуллером: 2 vCPU, 4 ГБ RAM, внешний накопитель типа SSD 120 ГБ, 500 Мб/с, 1200 IOPS, ОС Ubuntu 22.04.

Для сбора метрик и их отображения использовалось программное обеспечение *pgwatch2*[18], *Grafana*[19] и *PostgreSQL*, которое было запущено в докер-контейнере [13–15]. Под работоспособность системы выделялось 2 ГБ ОЗУ, 1 vCPU. Также в ходе тестирования увеличивалась производительность ВМ тестового стенда, поскольку некоторые тесты утилизировали все доступные ресурсы.

Тестирование проводилось с использованием нескольких сценариев подключения: напрямую к БД и через пуллер. Пуллер был установлен в режим сеансов. Сами сценарии включали в себя постепенное увеличение числа подключений (100, 500, 1000) и сложность запроса с размером активной сессии 10 мин.

Запросы носили следующий характер:

- простые запросы к пустой БД;
- crud-запросы (создание, чтение, модификация, удаление) с применением временных таблиц к БД, содержащей тестовые данные.

[15] vCPU – virtual central processing unit.

[16] RAM – random access memory.

[17] IOPS – input/output operations per second, количество операций ввода-вывода в секунду.

[18] PGWatch: Optimized PostgreSQL monitoring. https://pgwatch.com. Дата обращения 15.04.2023. / Accessed April 15, 2023.

[19] Grafana Labs. https://grafana.com. Дата обращения 12.04.2023. / Accessed April 12, 2023.

Для анализа выделялись следующие метрики:

1. Распределение использования ЦП:
   - idle – свободные ресурсы;
   - user – затраты на использование пользователями системы;
   - system – затраты на систему;
   - iowait – ожидание от дисковой подсистемы;
   - other – другие операции ЦП;
   - irqs – прерывания ядра ЦП.
2. Утилизация ЦП.
3. Утилизация ОЗУ.
4. Утилизация дисковой подсистемы (диска).

3.1. Запросы на чтение

В рамках данного тестирования параллельно выполнялся запрос на чтение к БД (получение версии СУБД).

Данный тест был выбран, т.к. позволяет максимально справедливо оценить влияние сессии на ресурсы БД (сильнее всего это отображается на утилизации ОЗУ).

Метрики при прямых подключениях простыми запросами изображены на рис. 4 (100 подключений), рис. 5 (500 подключений), рис. 6 (1000 подключений), где временные отрезки, показанные на графике,

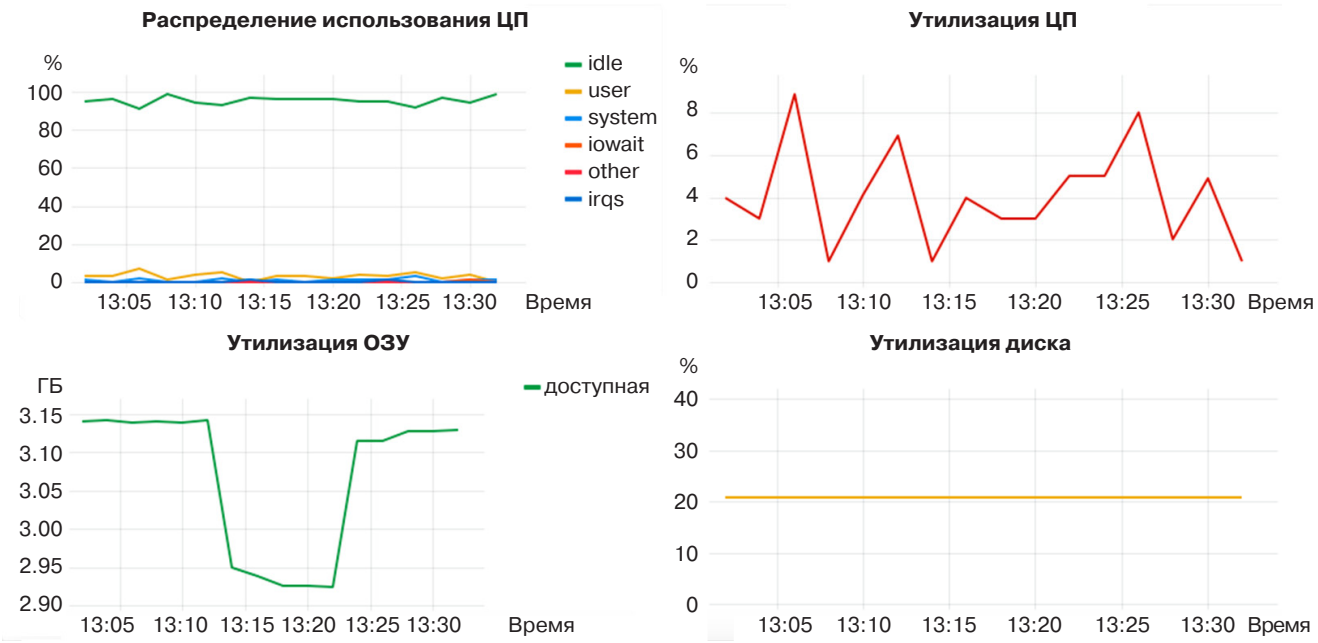

Рис. 4. Проведение нагрузочного тестирования простыми запросами с количеством подключений 100 напрямую к пустой БД

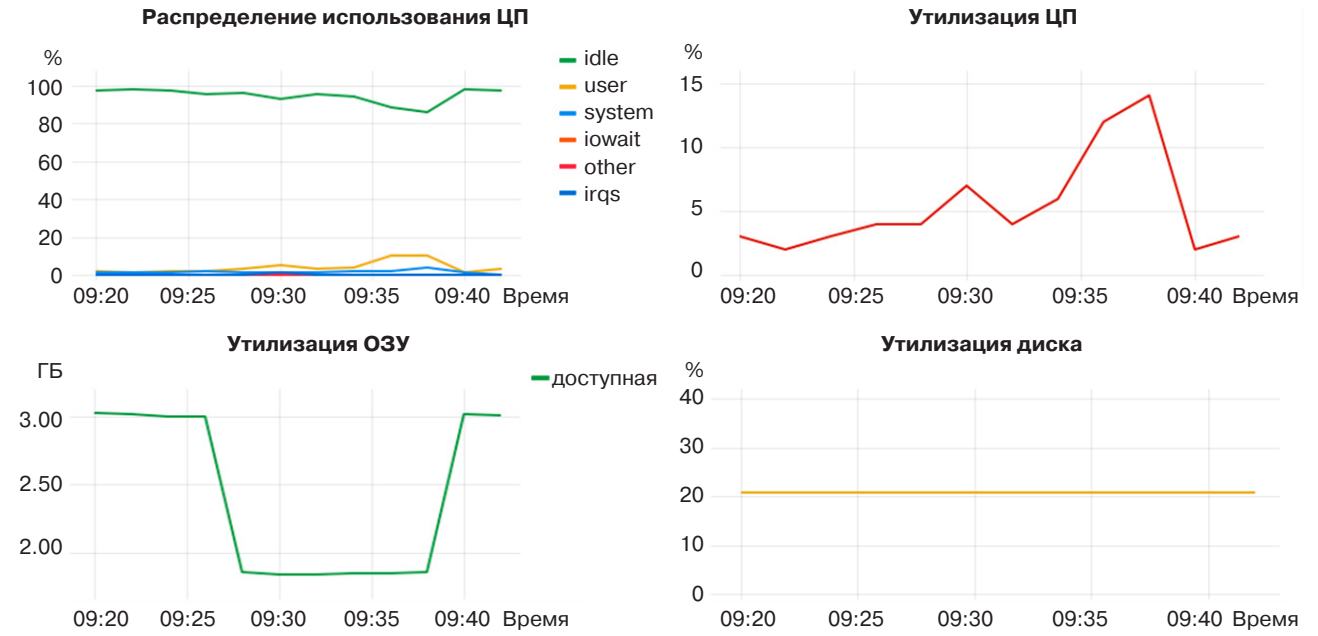

Рис. 5. Проведение нагрузочного тестирования простыми запросами с количеством подключений 500 напрямую к пустой БД

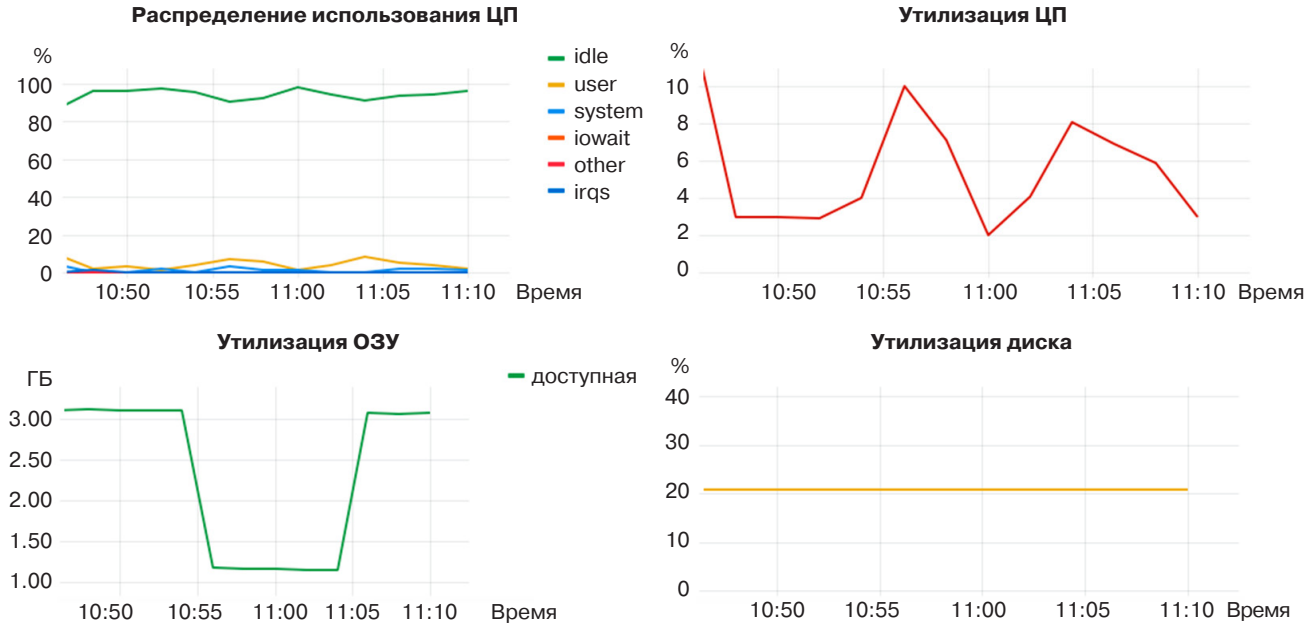

Рис. 6. Проведение нагрузочного тестирования простыми запросами с количеством подключений 1000 напрямую к пустой БД

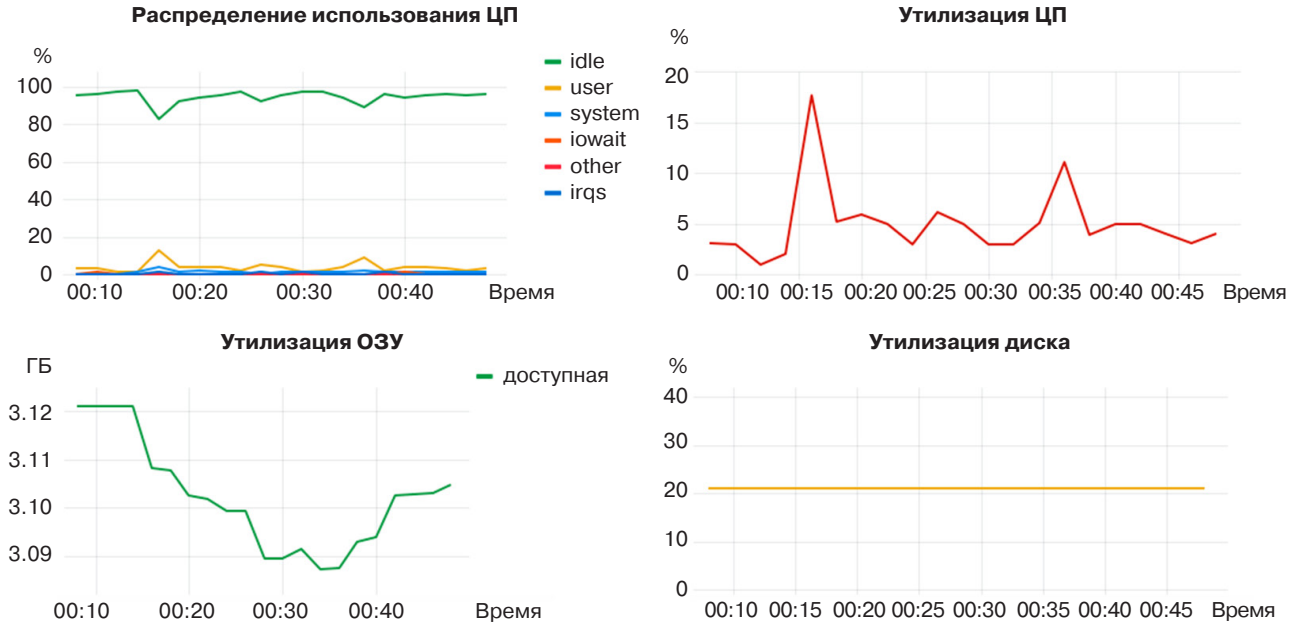

Рис. 7. Проведение нагрузочного тестирования простыми запросами с количеством подключений 100 через пуллер к пустой БД

отражают изменение параметров в период тестирования в реальном времени в формате «часы:минуты».

Метрики при подключениях через пуллер простыми запросами изображены на рис. 7 (100 подключений), рис. 8 (500 подключений), рис. 9 (1000 подключений).

В ходе тестирования ожидаемо было замечено сильное влияние простаивающих подключений на ресурсы, резервируемые под БД.

Если утилизацию ЦП и диска можно списать на погрешность и влияние внешних факторов, то ОЗУ стоит рассмотреть детальнее. Была получена следующая утилизация для тестов:

1. 100 подключений – 230 МБ (2.3 МБ/подключение);
2. 500 подключений – 1180 МБ (2.36 МБ/подключение);
3. 1000 подключений – 1810 МБ (1.81 МБ/подключение).

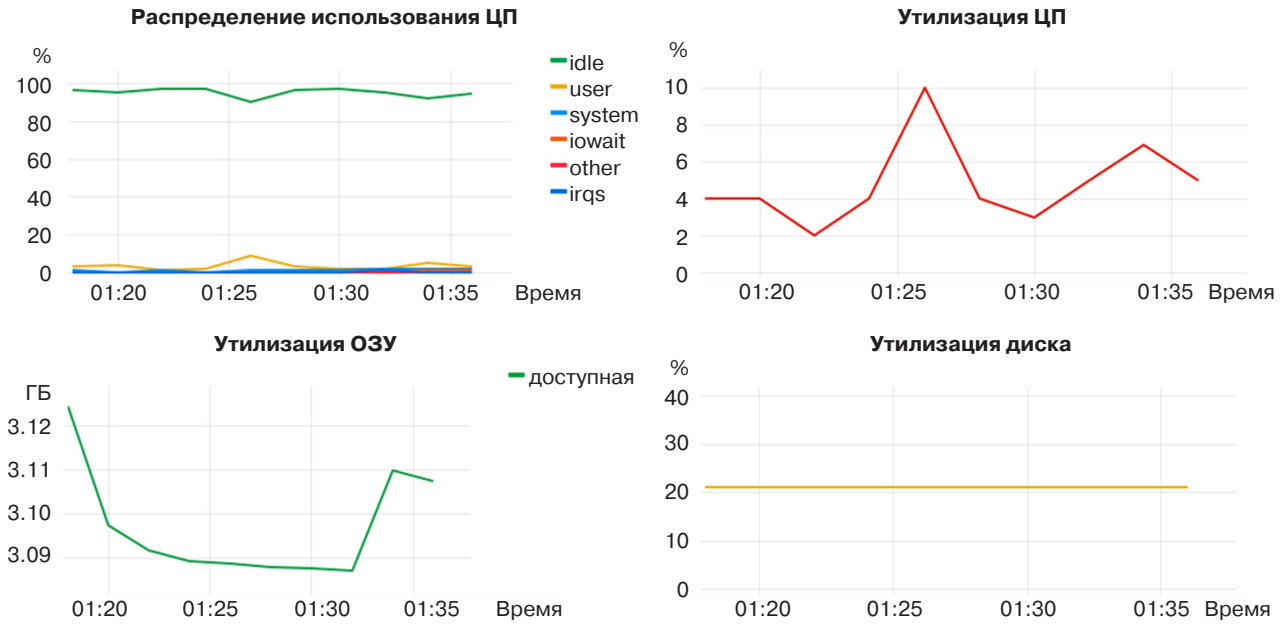

Рис. 8. Проведение нагрузочного тестирования простыми запросами с количеством подключений 500 через пуллер к пустой БД

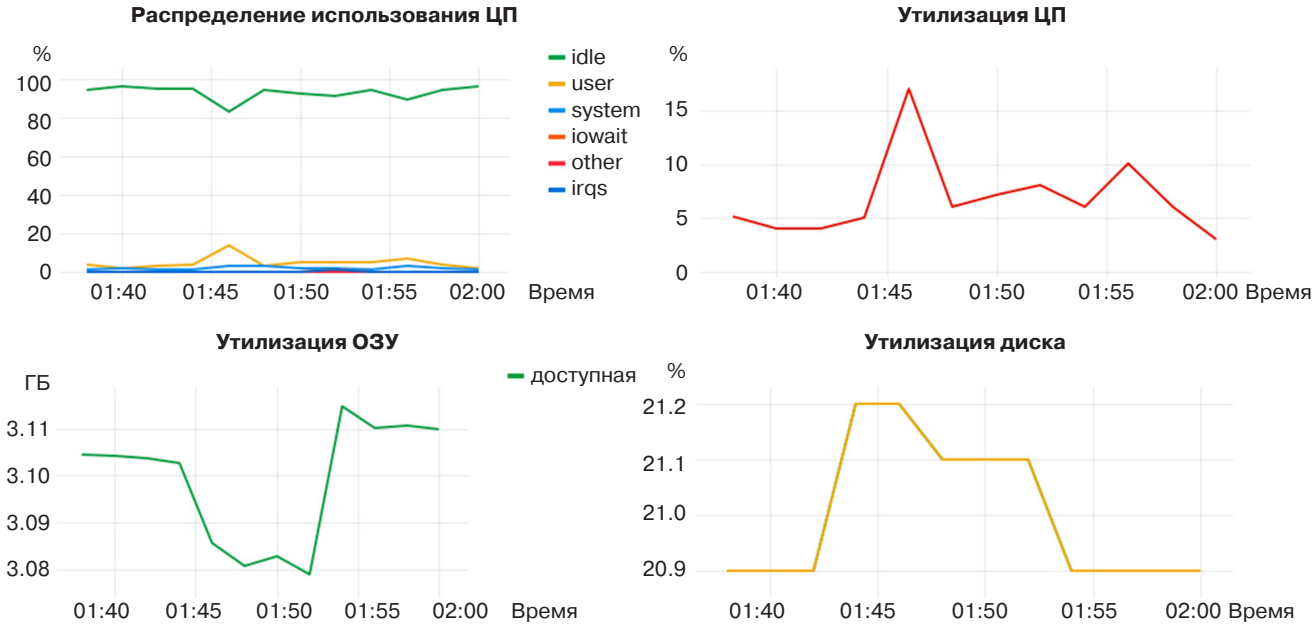

Рис. 9. Проведение нагрузочного тестирования простыми запросами с количеством подключений 1000 через пуллер к пустой БД

Показатели утилизации ОЗУ при тестировании через пуллер были крайне незначительны и не колебались в зависимости от количества подключений, оставаясь на крайне низком уровне (~30 МБ на всех тестах).

Также стоит отметить, что наибольшее влияние на потребляемые ресурсы единичное подключение оказывало при 500 параллельных подключений.

3.2. Усложненные запросы

В рамках данного тестирования параллельно выполнялся запрос на запись к БД, создание, наполнение, удаление таблицы (2 столбца, 1000000 рядов, с типом значения «text»).

Данный тест был выбран, т.к. позволяет оценить возможную экономию ресурсов ЦП и диска при использовании пуллера запросов.

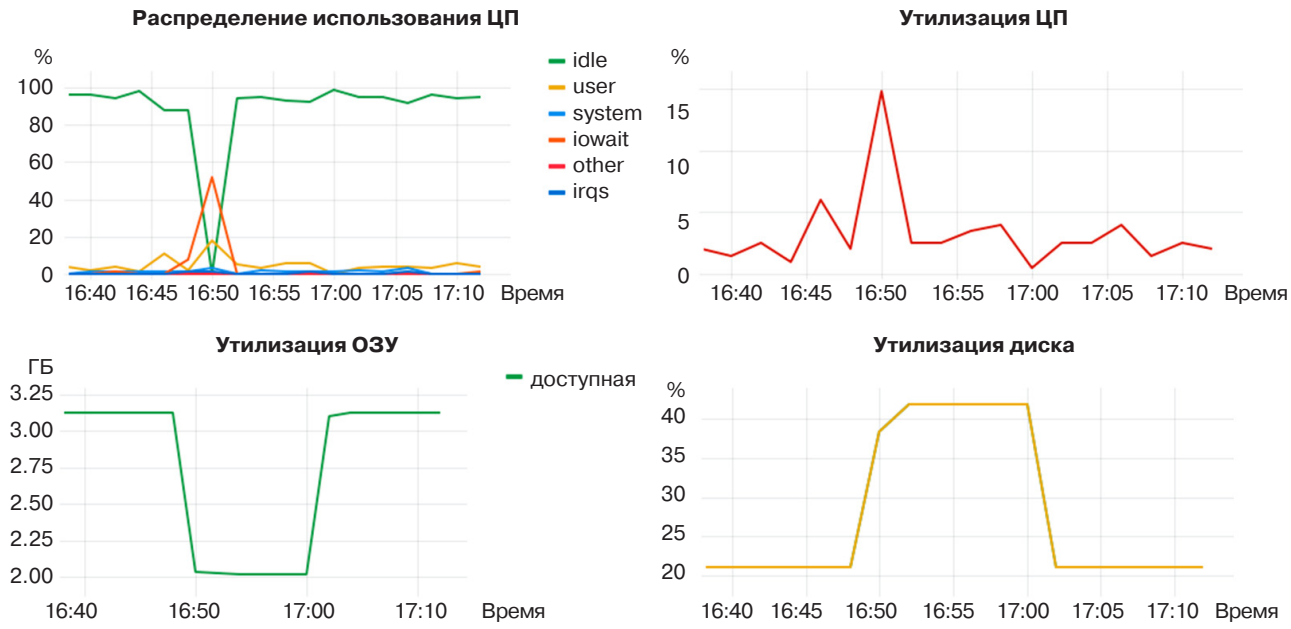

Рис. 10. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 100 к БД, заполненной тестовыми данными

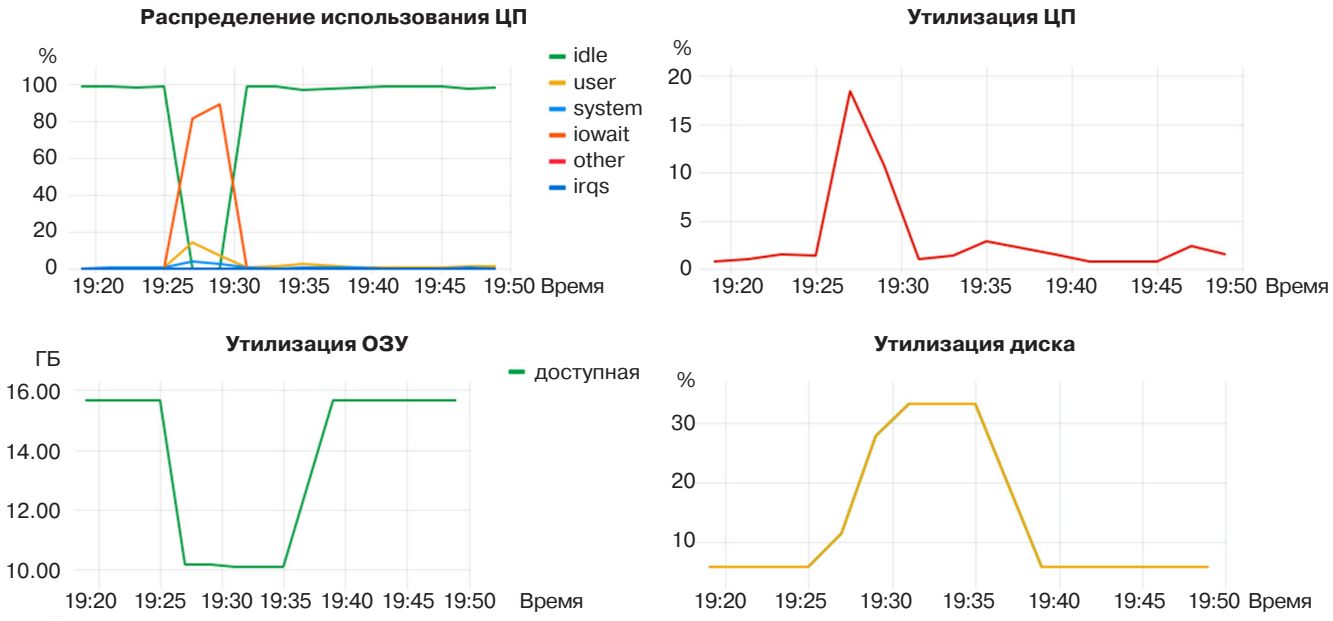

Рис. 11. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 500 к БД, заполненной тестовыми данными

Метрики при прямых подключениях усложненными запросами изображены на рис. 10 (100 подключений), рис. 11 (500 подключений), рис. 12 (1000 подключений), где временные отрезки, показанные на графике, отражают изменение параметров в период тестирования в реальном времени в формате «часы:минуты».

Метрики при подключениях через пуллер усложненными запросами изображены на рис. 13 (100 подключений), рис. 14 (500 подключений), рис. 15 (1000 подключений).

Тестирование показало большое влияние параллельных операций на ресурсы БД и продемонстрировало возможность минимизировать их средствами пуллера.

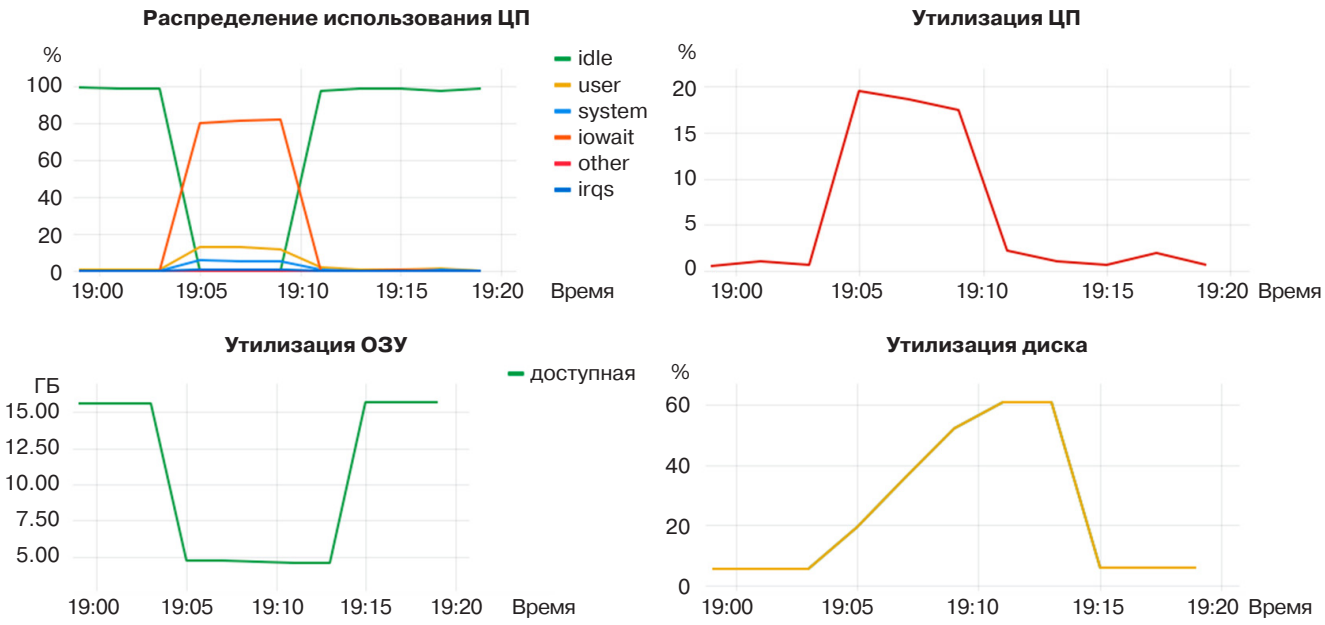

Рис. 12. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 1000 к БД, заполненной тестовыми данными

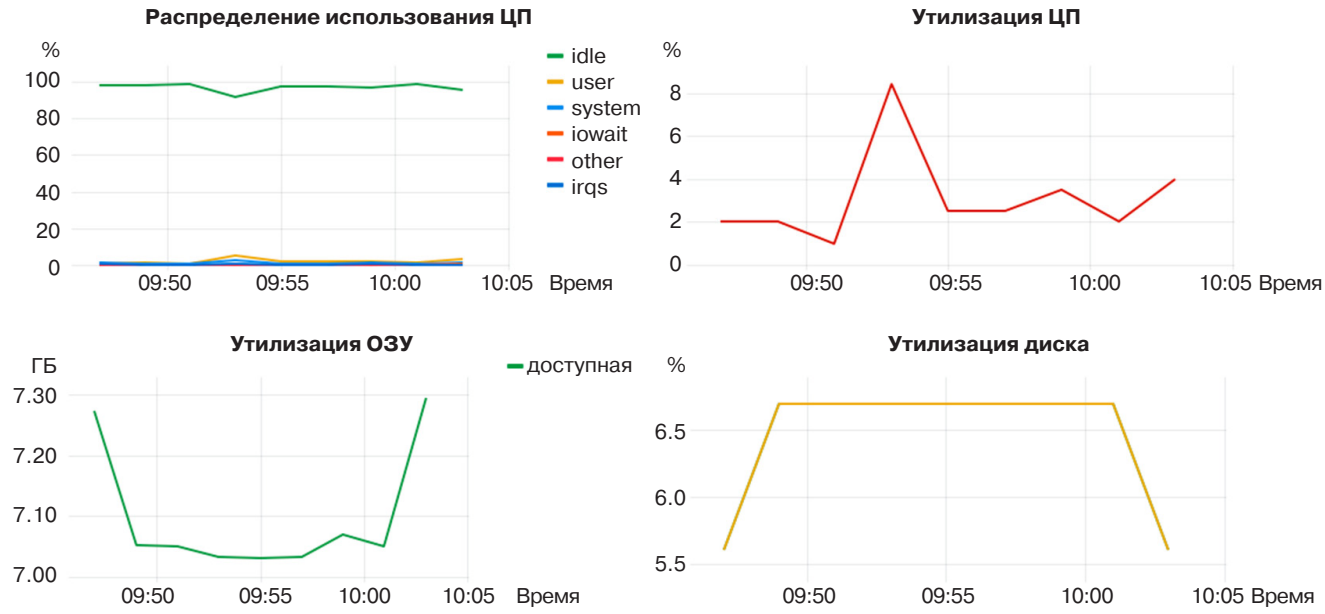

Рис. 13. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 100 через пуллер к БД, заполненной тестовыми данными

Рассмотрим утилизацию ОЗУ при разном количестве подключений напрямую к БД:

1. 100 подключений – 1125 МБ (11.25 МБ/подключение);
2. 500 подключений – 5900 МБ (11.8 МБ/подключение);
3. 1000 подключений – 10250 МБ (10.25 МБ/подключение).

Показатели утилизации ОЗУ при тестировании через пуллер были крайне незначительны и не колебались в зависимости от количества подключений, оставаясь на крайне низком уровне (~250 МБ на всех тестах).

В сравнении с предыдущим тестом видны скачки утилизации по всем отслеживаемым метрикам. Тест на 500 подключений также, как и в прошлом случае, показывает самую высокую утилизацию памяти

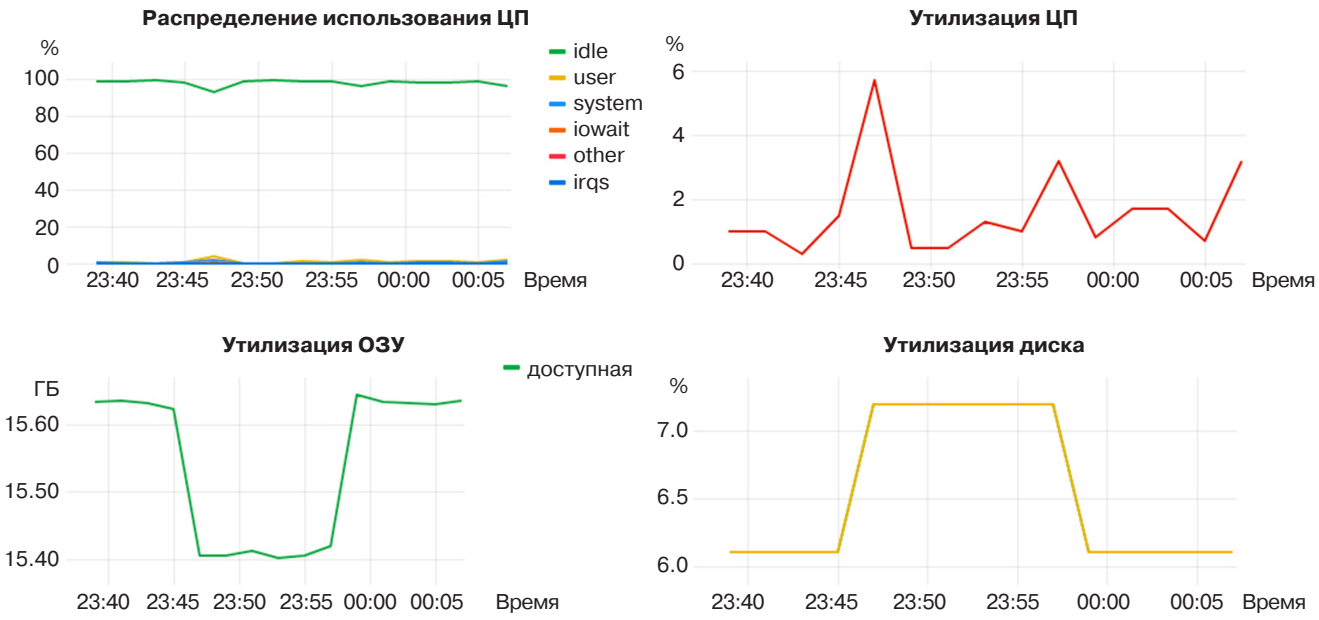

Рис. 14. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 500 через пуллер к БД, заполненной тестовыми данными

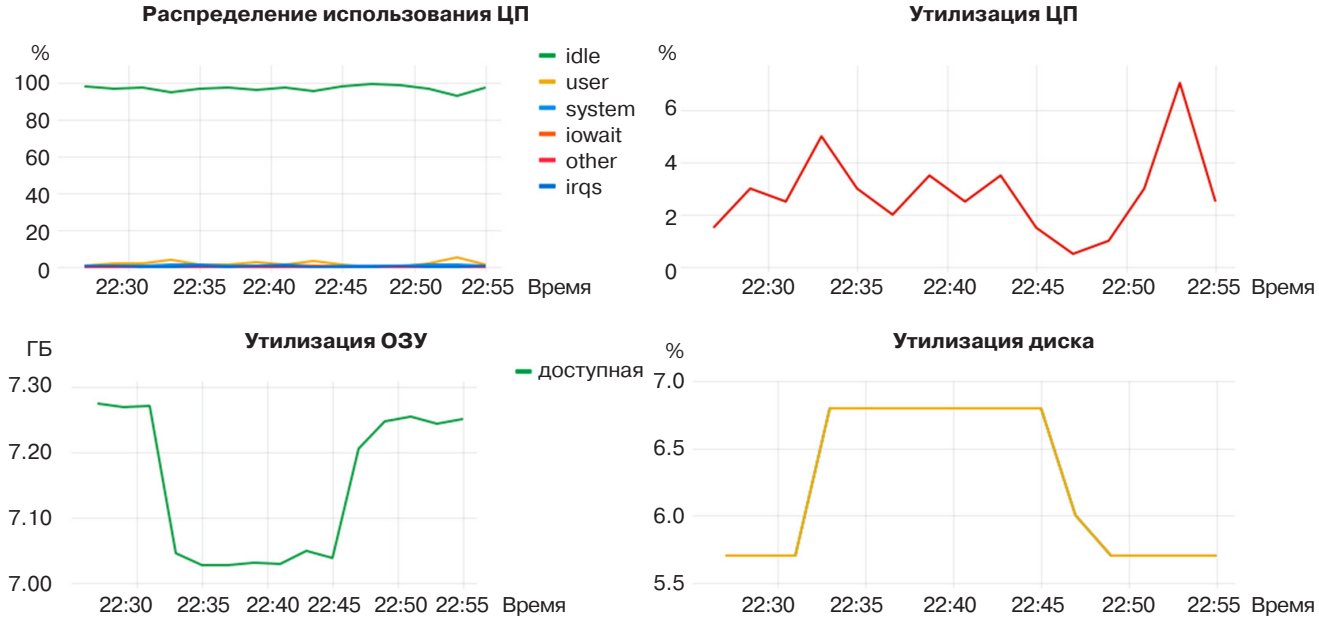

Рис. 15. Проведение нагрузочного тестирования усложненными запросами с использованием временных таблиц и количеством подключений 1000 через пуллер к БД, заполненной тестовыми данными

на единичное подключение. Тест на 1000 подключений показал, что СУБД не ориентирована на такое количество подключений, время повышенной утилизации ЦП и дисковой системы сравнительно выше остальных случаев.

Также можно заметить оптимизацию утилизации дисковой подсистемы. Это связано с возможностью выноса обработки операций с временными таблицами на мощности пуллера (он заранее знает результат работы всех запросов).

Уменьшение загрузки ЦП при тестировании через пуллер связано с тем, что он берет управление сессиями на себя (самый затратный момент по ресурсам ЦП).

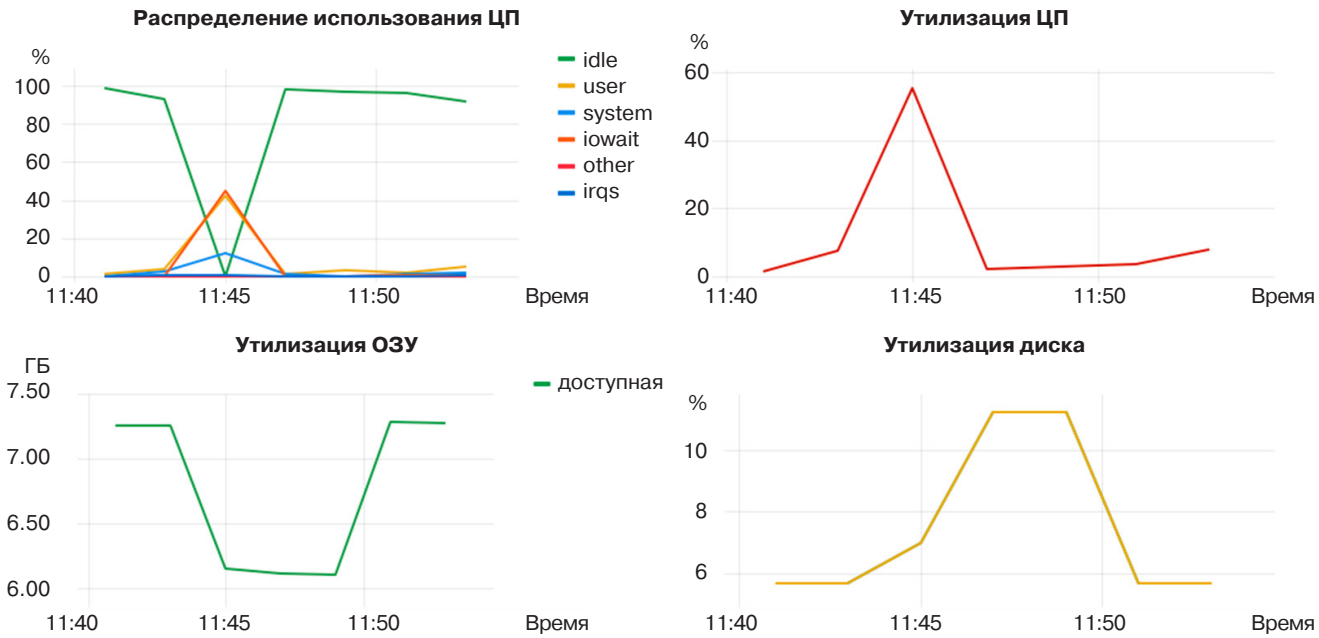

Рис. 16. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 100 напрямую к БД

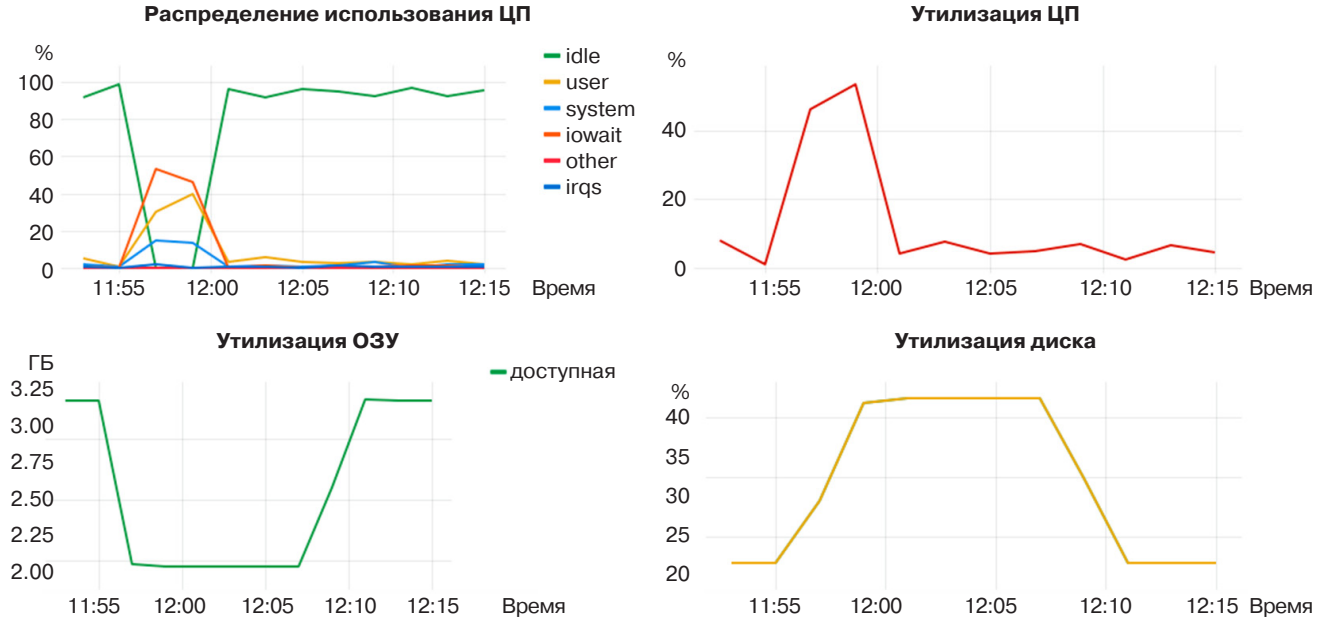

Рис. 17. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 500 напрямую к БД

3.3. Комбинированные запросы

Данный тест является комбинацией двух предыдущих. Он направлен на оценку более общего случая использования БД (параллельная запись и чтение).

Метрики при прямых подключениях комбинированными запросами изображены на рис. 16 (100 подключений), рис. 17 (500 подключений), рис. 18 (1000 подключений), где временные отрезки, показанные на графике, отражают изменение параметров в период тестирования в реальном времени в формате «часы:минуты».

Метрики при подключениях через пуллер комбинированными запросами изображены на рис. 19 (100 подключений), рис. 20 (500 подключений), рис. 21 (1000 подключений).

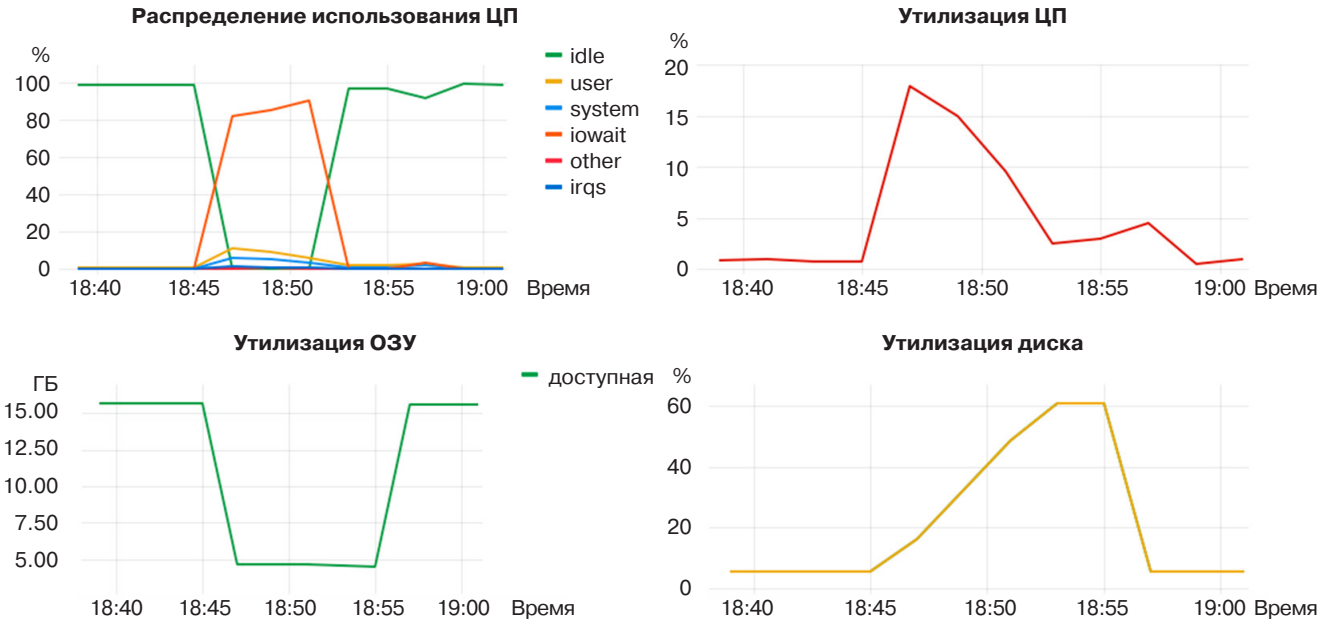

Рис. 18. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 1000 напрямую к БД

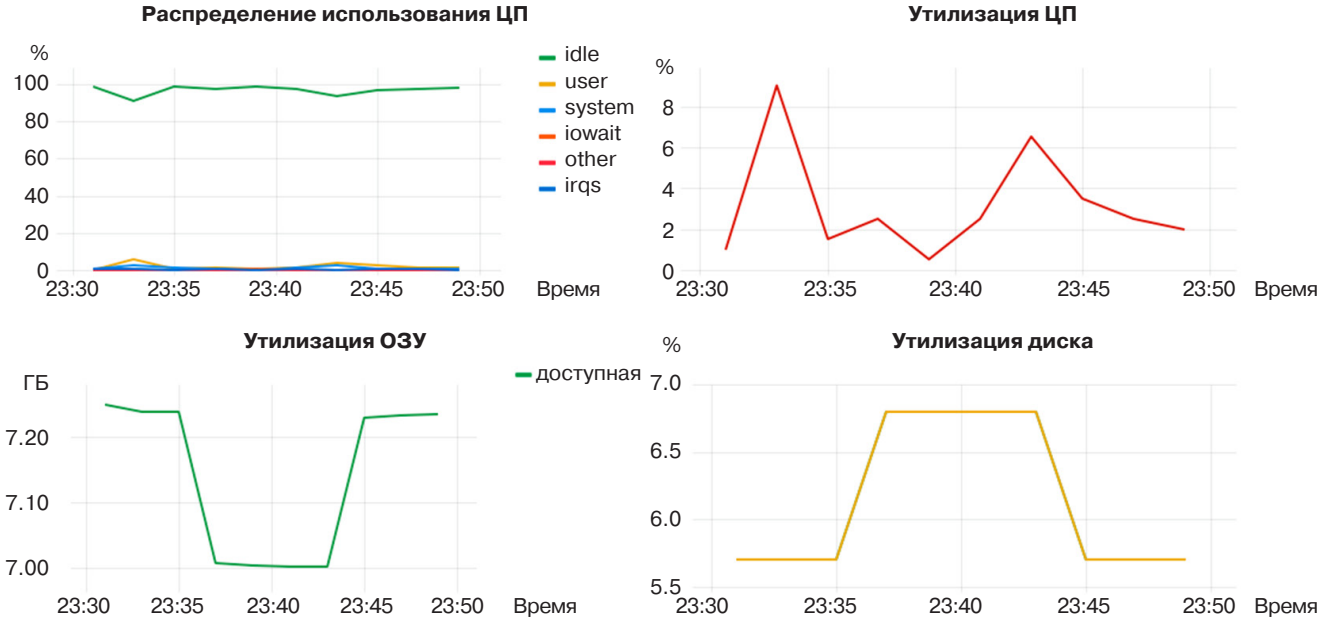

Рис. 19. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 100 через пуллер к БД

Тестирование не показало никаких аномалий при проведении тестов на чтение и запись одновременно. Это показывает, что тестирование с более реальными случаями никак не конфликтует с использованием балансировщика подключений.

Рассмотрим утилизацию ОЗУ при разном количестве подключений напрямую к БД:

1. 100 подключений – 1200 МБ (12.0 МБ/подключение);
2. 500 подключений – 6050 МБ (12.1 МБ/подключение);
3. 1000 подключений – 11150 МБ (11.15 МБ/подключение).

Показатели утилизации ОЗУ при тестировании через пуллер были крайне незначительны и не колебались в зависимости от количества подключений, оставаясь на крайне низком уровне (~250 МБ на всех тестах).

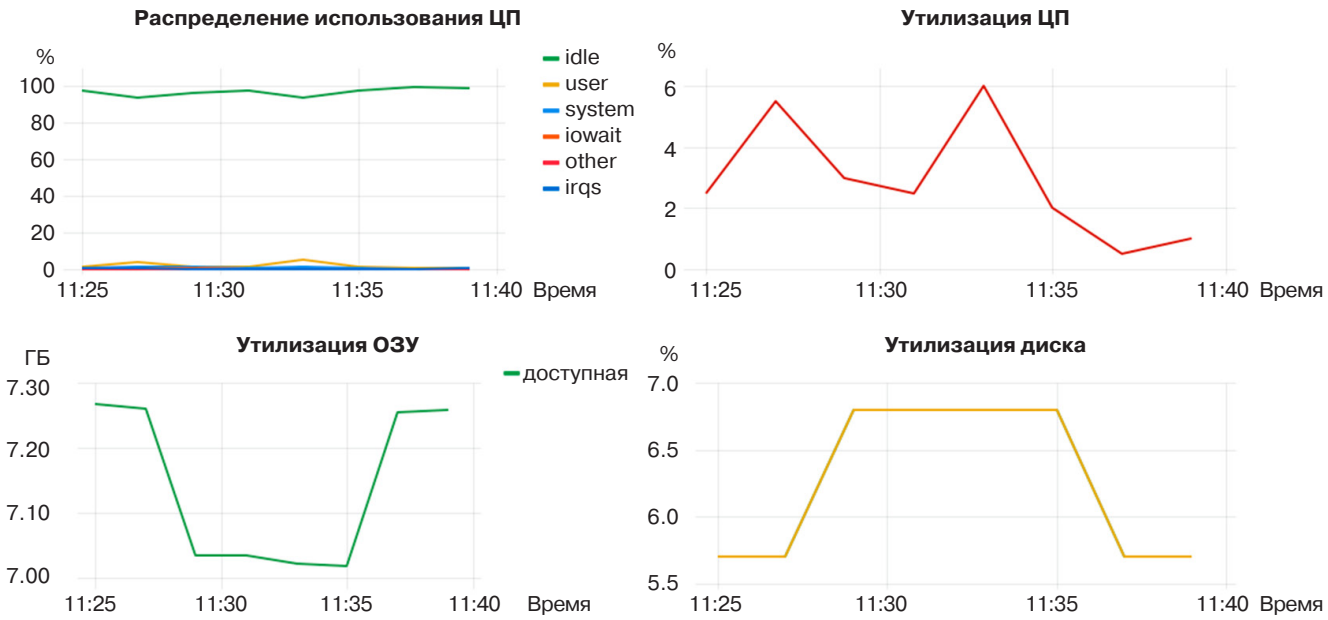

Рис. 20. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 500 через пуллер к БД

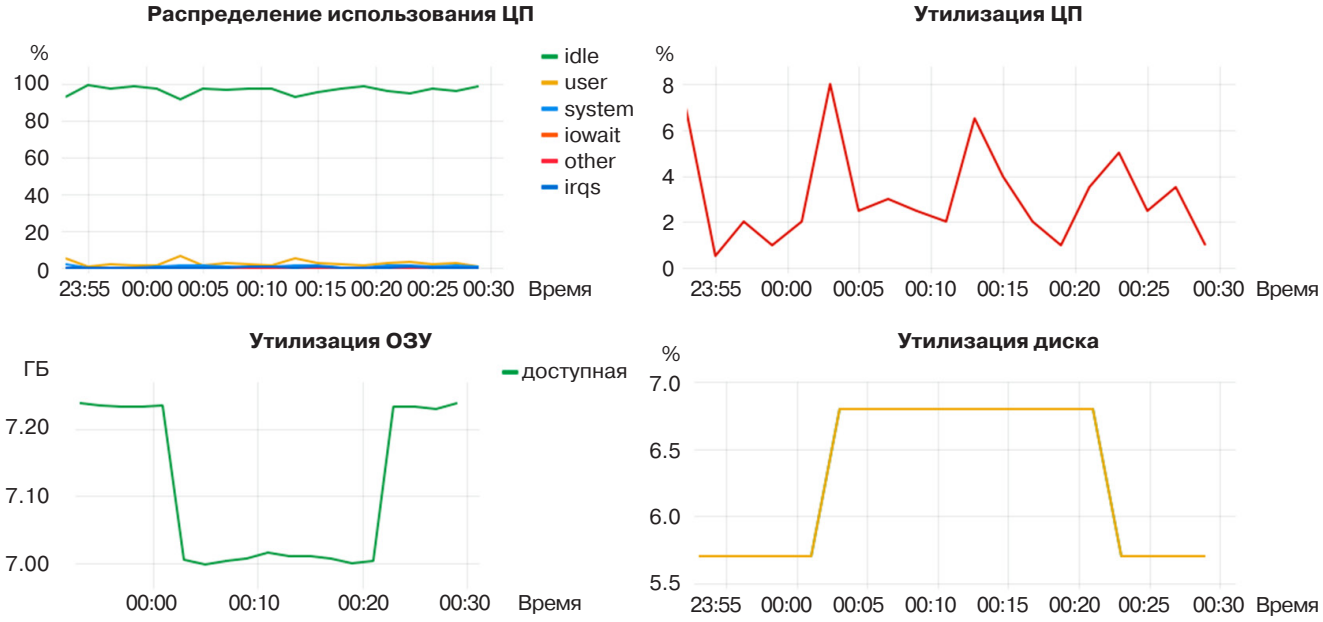

Рис. 21. Проведение нагрузочного тестирования комбинированными запросами с использованием временных таблиц и количеством подключений 1000 через пуллер к БД

3.4. Общая оценка результатов

В целом тестирование показало ожидаемые результаты, а именно снижение нагрузки на систему. Таким образом, использование балансировщика позволит не только снизить затраты на инфраструктуру, но и сильно оптимизировать саму систему, в особенности избежать «огромных» ВМ по несколько сотен гигабайт ОЗУ и ядер ЦП под СУБД. Тестирование показало, что в среднем оптимизация подключений позволила освободить 25–50% ОЗУ, которые были предназначены для самой СУБД, учитывая размер ВМ, выделяемой под сам пуллер. Если же взять чистые вычисления, без учета ресурсов под сам балансировщик, то разница составила ~30 раз. При этом на разных тестах *PgBouncer* показывал примерно одни и те же значения, что свидетельствует о некоторой универсальности данного решения в отличии от корректировки параметров самой СУБД.

Таблица 2. Результаты нагрузочного тестирования простыми запросами

Количество запросов	Способ отправки запросов к БД	Утилизация ЦП, %	Утилизация ОЗУ, МБ	Утилизация диска, %
100 запросов	Напрямую к БД	7	230	0
	Через пуллер к БД	14	30	0
500 запросов	Напрямую к БД	11	1100	0
	Через пуллер к БД	8	30	0
1000 запросов	Напрямую к БД	8	1800	0
	Через пуллер к БД	11	20	1.4

Таблица 3. Результаты нагрузочного тестирования усложненными запросами

Количество запросов	Способ отправки запросов к БД	Утилизация ЦП, %	Утилизация ОЗУ, МБ	Утилизация диска, %
100 запросов	Напрямую к БД	25	1125	20
	Через пуллер к БД	8	260	1.5
500 запросов	Напрямую к БД	16	5900	30
	Через пуллер к БД	6	250	1.5
1000 запросов	Напрямую к БД	19	10250	55
	Через пуллер к БД	6	250	1.1

Таблица 4. Результаты нагрузочного тестирования комбинированными запросами

Количество запросов	Способ отправки запросов к БД	Утилизация ЦП, %	Утилизация ОЗУ, МБ	Утилизация диска, %
100 запросов	Напрямую к БД	53	1200	7
	Через пуллер к БД	8	250	1
500 запросов	Напрямую к БД	58	6050	32
	Через пуллер к БД	4	260	1
1000 запросов	Напрямую к БД	18	11150	55
	Через пуллер к БД	7	250	1

Стоит отметить снижение утилизации ресурсов дисковой подсистемы при использовании балансировщиков запросов. Данная оптимизация также позволит снизить затраты и уменьшить параметры диска, предназначенного под БД.

Также было замечено снижение утилизации ЦП при использовании пуллера. В среднем колебание составило 15–20%, что трудно назвать оптимизацией, поскольку нагрузка на ЦП имеет моментальный, пиковый характер, а система мониторинга собирает данные 1 раз в минуту, и, следовательно, могут наблюдаться резкие скачки показаний. Полученные результаты можно учитывать при проектировании системы.

Приведенные на рисунках графики «Распределение использования ЦП» при тестировании отображают информацию о том, что ресурсы непосредственно затрачиваются для работы СУБД *PostgreSQL*, а не для других процессов, например, iowait, который возникает при максимальной нагрузке на дисковую подсистему.

Сводные данные проведенного тестирования представлены в табл. 2–4.

ЗАКЛЮЧЕНИЕ

Исследование показало, что программное обеспечение *PgBouncer* представляет собой эффективный инструмент для управления пулом соединений с БД *PostgreSQL*. В ходе проведенного тестирования было выявлено улучшение производительности системы путем снижения затрачиваемых ресурсов на СУБД *PostgreSQL*, а именно нагрузка на ЦП уменьшилась на 15%, на ОЗУ – на 25–50%, на дисковую подсистему – на 20%.

Данный балансировщик имеет гибкую и легкую систему настройки режимов работы, позволяет выбрать наиболее подходящий вариант в зависимости от конкретных потребностей приложений и настроек БД.

Применение *PgBouncer* повышает надежность БД и сокращает время обработки запросов. Это особенно важно для приложений, работающих с большим объемом данных, которые обрабатывают множество запросов одновременно.

Таким образом, можно сделать вывод о том, что *PgBouncer* является полезным инструментом для

управления БД *PostgreSQL* и может быть успешно применен во многих приложениях и инфраструктурах, где требуется высокая производительность, масштабируемость и безопасность.

В дальнейшем планируется изучение способов развертывания данного архитектурного решения на базе российской платформы и проведение нагрузочного тестирования для определения целесообразности переноса систем, а также проведение тестирования в составе кластера высокой доступности.

Вклад авторов

А.С. Боронников – идея исследования, проведение исследования, написание текста статьи, интерпретация и обобщение результатов исследования, научное редактирование статьи.

П.С. Цынгалёв – консультации по вопросам проведения исследования, написание текста статьи.

В.Г. Ильин – консультации по вопросам проведения исследования, написание текста статьи.

Т.А. Деменкова – идея исследования, планирование исследования, научное редактирование статьи.

Authors' contributions

A.S. Boronnikov – the research idea, conducting research, writing the text of the article, interpretation and generalization of the results of the research, scientific editing of the article.

P.S. Tsyngalev – consultations on research issues, writing the text of the article.

V.G. Ilyin – consultations on research issues, writing the text of the article.

T.A. Demenkova – the research idea, research planning, scientific editing of the article.

СПИСОК ЛИТЕРАТУРЫ

1. Шараев Е.В. Использование алгоритмических композиций при оптимизации PostgreSQL методами машинного обучения. *Научному Прогрессу – Творчество Молодых*. 2019;3:135–137.
2. Borodin A., Mirvoda S., Porshnev S., Kulikov I. Optimization of Memory Operations in Generalized Search Trees of PostgreSQL. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (Eds.). Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation. BDAS 2017. *Communications in Computer and Information Science*. 2017;716:224–232. https://doi.org/10.1007/978-3-319-58274-0_19
3. Варакута П.С., Козлов Р.К. Имитационное моделирование пропускной способности пулов соединений к базе данных PostgreSQL. *Трибуна ученого*. 2022;5:48–53.
4. Мухамедина А., Айдаров А.К. Современные средства балансировки перегрузки. *Научные исследования XXI века*. 2021;2:105–109.
5. Gudilin D.S., Zvonarev A.E., Goryachkin B.S., Lychagin D.A. Relational Database Performance Comparation. In: *Proc. 5th International Youth Conference on Radio Electronics, Electrical and Power Engineering* (*REEPE*). March, 16–18, 2023. Moscow. https://doi.org/10.1109/REEPE57272.2023.10086872
6. Тупикина М.А. Сравнение систем управления базами данных SQLite, MySQL и PostgreSQL. В сб.: *Студенческая наука для развития информационного общества: сборник материалов VIII Всероссийской научно-технической конференции*. Часть 2; 22–23 мая 2018 г. Ставрополь: Северо-Кавказский федеральный университет; 2018. С. 345–347.
7. Виноградова М.В., Барашкова Е.С., Березин И.С., Ореликов М.Г., Лузин Д.С. Обзор системы полнотекстового поиска в постреляционной базе данных PostgreSQL. *E-SCIO*. 2020;5(44):754–778.
8. Пантилимонов М.В., Бучацкий Р.А., Жуйков Р.А. Кэширование машинного кода в динамическом компиляторе SQL-запросов для СУБД PostgreSQL. *Труды Института системного программирования РАН*. 2020;32(1):205–220. https://doi.org/10.15514/ISPRAS-2020-32(1)-11
9. Портретов В.С. Сравнение PostgreSQL и MySQL. *Молодежная наука в развитии регионов*. 2017;1:136–139.
10. Chauhan C., Kumar D. *PostgreSQL High Performance Cookbook*. 2nd ed. Birmingham: Packt Publishing; 2017. 360 p.
11. Рогов Е.В. *PostgreSQL 15 изнутри*. М.: ДМК Пресс; 2023. 662 с.
12. Новиков Б.А., Горшкова Е.А., Графеева Н.Г. *Основы технологий баз данных*. 2-е изд. М.: ДМК Пресс; 2020. 582 с.
13. Бойченко А.В., Рогожин Д.К, Корнеев Д.Г. Алгоритм динамического масштабирования реляционных баз данных в облачных средах. *Статистика и Экономика.* 2014;6–2:461–465.
14. Афанасьев Г.И., Абулкасимов М.М., Белоногов И.Б. Методика создания Docker-образа PostgreSQL в среде Ubuntu Linux. *Аллея науки.* 2018;2(1–17):913–918.
15. Smolinski M. Impact of storage space configuration on transaction processing performance for relational database in PostgreSQL. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (Eds.). Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation. BDAS 2017. *Communications in Computer and Information Science*. 2018;928:157–167. https://doi.org/10.1007/978-3-319-99987-6_12

REFERENCES

1. Sharaev E.V. Using Algorithmic Compositions in PostgreSQL Optimization with Machine Learning Methods. *Nauchnomu Progressu – Tvorchestvo Molodykh*. 2019;3:135–137 (in Russ.).

2. Borodin A., Mirvoda S., Porshnev S., Kulikov I. Optimization of Memory Operations in Generalized Search Trees of PostgreSQL. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (Eds.). Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation. BDAS 2017. *Communications in Computer and Information Science*. 2017;716:224–232. https://doi.org/10.1007/978-3-319-58274-0_19
3. Varakuta P.S., Kozlov R.K. Simulation of the capacity of connection pools to the PostgreSQL database. *Tribuna uchenogo = Tribune of the Scientist*. 2022;5:48–53 (in Russ.).
4. Mukhamedina A., Aidarov A.K. Modern load balancing tools. *Nauchnye issledovaniya 21 veka = Scientific Research of the 21st Century*. 2021;2:105–109 (in Russ.).
5. Gudilin D.S., Zvonarev A.E., Goryachkin B.S., Lychagin D.A. Relational Database Performance Comparation. In: *Proc. 5th International Youth Conference on Radio Electronics, Electrical and Power Engineering* (*REEPE*). March 16–18, 2023. Moscow. https://doi.org/10.1109/REEPE57272.2023.10086872
6. Tupikina M.A. Comparison of database management systems SQLite, MySQL and PostgreSQL. In: *Student Science for the Development of the Information Society: collection of materials of the 8th All-Russian Scientific and Technical Conference. Part 2*; May 22–23, 2018. Stavropol: North Caucasian Federal University; 2018. P. 345–347 (in Russ.).
7. Vinogradova M.V., Barashkova E.S., Berezin I.S., Orelikov M.G., Luzin D.S. An overview of the full-text search system in PostgreSQL post-relational database. *E-SCIO*. 2020;5(44):754–778 (in Russ.).
8. Pantilimonov M.V., Buchatskiy R.A., Zhuykov R.A. Machine code caching in PostgreSQL query JIT-compiler. *Trudy Instituta sistemnogo programmirovaniya RAN = Proceedings of the Institute for System Programming of the RAS* (*Proceedings of ISP RAS*). 2020;32(1):205–220 (in Russ.).
9. Portretov V.S. Comparison of PostgreSQL and MySQL. *Molodezhnaya Nauka v Razvitii Regionov*. 2017;1:136–139 (in Russ.).
10. Chauhan C., Kumar D. *PostgreSQL High Performance Cookbook*. 2nd ed. Birmingham: Packt Publishing; 2017. 360 p.
11. Rogov E.V. *PostgreSQL 15 iznutri* (*PostgreSQL 15 from the Inside*). Moscow: DMK Press; 2023. 662 p. (in Russ.).
12. Novikov B.A., Gorshkova E.A., Grafeeva N.G. *Osnovy tekhnologii baz dannykh* (*Bases of Technologies of Databases*). 2nd ed. Moscow: DMK Press; 2020. 582 p. (in Russ.).
13. Boichenko A.V., Rogojin D.K., Korneev D.G. Algorithm for dynamic scaling relational database in clouds. *Statistika i Ekonomika = Statistics and Economics.* 2014;6–2:461–465 (in Russ.).
14. Afanas'ev G.I., Abulkasimov M.M., Belonogov I.B. How to create a PostgreSQL Docker image on Ubuntu Linux. *Alleya nauki = Alley of Science*. 2018;2(1–17):913–918 (in Russ.).
15. Smolinski M. Impact of storage space configuration on transaction processing performance for relational database in PostgreSQL. In: Kozielski S., Mrozek D., Kasprowski P., Małysiak-Mrozek B., Kostrzewa D. (Eds.). Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation. BDAS 2017. *Communications in Computer and Information Science*. 2018;928:157–167. https://doi.org/10.1007/978-3-319-99987-6_12

Об авторах

Боронников Антон Сергеевич, аспирант, старший преподаватель, кафедра вычислительной техники, Институт информационных технологий, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: boronnikov-anton@mail.ru. SPIN-код РИНЦ 8232-6328, https://orcid.org/0009-0008-4911-6609

Цынгалёв Павел Сергеевич, студент, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: pstsyngalev@mail.ru. https://orcid.org/0009-0007-6354-1364

Ильин Виктор Георгиевич, студент, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: vgilyin@yahoo.com. https://orcid.org/0009-0001-0304-3052

Деменкова Татьяна Александровна, к.т.н., доцент, кафедра вычислительной техники, Институт информационных технологий, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: demenkova@mirea.ru. Scopus Author ID 57192958412, ResearcherID AAB-3937-2020, SPIN-код РИНЦ 3424-7489, https://orcid.org/0000-0003-3519-6683

About the authors

Anton S. Boronnikov, Postgraduate Student, Senior Lecture, Computer Engineering Department, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: boronnikov-anton@mail.ru. RSCI SPIN-code 8232-6328, https://orcid.org/0009-0008-4911-6609

Pavel S. Tsyngalev, Student, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: pstsyngalev@mail.ru. https://orcid.org/0009-0007-6354-1364

Victor G. Ilyin, Student, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: vgilyin@yahoo.com. https://orcid.org/0009-0001-0304-3052

Tatiana A. Demenkova, Cand. Sci. (Eng.), Associate Professor, Computer Engineering Department, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: demenkova@mirea.ru. Scopus Author ID 57192958412, ResearcherID AAB-3937-2020, RSCI SPIN-code 3424-7489, https://orcid.org/0000-0003-3519-6683