Моделирование работы пользователей с многосерверной базой данных

Е.Г. Агапова, Г.Г. Немцов

Тихоокеанский государственный университет, Хабаровск

Аннотация: В работе рассмотрено моделирование работы пользователей с многосерверной базой данных, разработанной на основе микросервисной архитектуры. Был проведён анализ предметной области, описаны основные сущности системы, а также реализованы механизмы передачи данных и взаимодействия сервисов с использованием Docker и Apache Kafka. Выявлено, что разработка многосерверной базы данных позволила добиться высокой масштабируемости и отказоустойчивости системы. Реализация механизмов репликации и шардирования обеспечила равномерное распределение нагрузки, а использование брокера сообщений Kafka способствовало эффективному обмену данными между сервисами. Проведённое тестирование подтвердило надёжность системы при высокой нагрузке, а также выявило её сильные стороны и потенциальные улучшения.

Ключевые слова: моделирование, балансировка нагрузки, Docker, Apache Kafka, микросервисная архитектура, распределённые системы, оптимизация запросов.

Введение

Современные информационные технологии активно развиваются в направлении повышения производительности и масштабируемости систем хранения данных. С ростом объёма данных, генерируемых различными источниками, в том числе датчиками интернета вещей (IoT), необходимость в надёжных, отказоустойчивых и высокопроизводительных решениях становится всё более очевидной [1]. В данном контексте многосерверные базы данных, интегрированные с такими инструментами, как Docker и Apache Kafka, предоставляют гибкие возможности для построения современных распределённых систем [2, 3].

Docker позволяет запускать несколько контейнеров на одном сервере, создавая виртуальные изолированные среды, которые имитируют работу нескольких серверов. Это упрощает процесс развертывания, тестирования и масштабирования приложений. В свою очередь, Apache Kafka выполняет роль брокера сообщений, обеспечивая надёжный и быстрый обмен данными между микросервисами. Сочетание этих технологий позволяет достигать

высокой производительности и отказоустойчивости в системах обработки данных.

В данной статье исследуется процесс моделирования работы пользователей с многосерверной базой данных. Основное внимание уделяется разработке микросервисной архитектуры, построенной с использованием Docker для изоляции компонентов системы, и применению Apache Kafka для передачи данных. Для имитации работы пользователей используется система, которая собирает данные о погодных условиях через датчики и передаёт их на сервер для последующей обработки и анализа.

Анализ научной литературы

Архитектуры микросервисов и сервисные сетки стали очень популярными и сталкиваются со все более строгими требованиями к масштабируемости и надежности. Чтобы добиться выполнения сервисов с низкой задержкой и максимизировать производительность, поставщики услуг крупномасштабных распределенных систем развертывают микросервисы географически ближе к своим пользователям в многокластерных сервисных сетках. Однако межкластерные зависимости сервисов вносят дополнительную задержку, и эффективная балансировка нагрузки между несколькими репликами, распределенными по кластерам, имеет решающее значение. Решая эту проблему, мы представляем L3, адаптивный механизм балансировки нагрузки с учетом задержек для многокластерных сервисных сеток [4].

В научных исследованиях предложены архитектуры приложения с микросервисным подходом и приведен сравнительный анализ [5]. Бондаренко А. С., Королев Д. В., и Зайцев К. С. в своем исследованияи построили архитектуру приложения с микросервисным подходом для диагностики заболеваний щитовидной железы по снимкам УЗИ [6]. В работе [7] авторы сформировали открытую, многоуровневую,

сегментированную, сервис–ориентированную архитектуру программного комплекса, основанную на использовании технологий микросервисов, контейнерной виртуализации и обмена сообщениями по подписке. В [8] приведено описание архитектурных решений для современных Enterprise-системы, работающих под большой и постоянно возрастающей нагрузкой: обработка и хранение больших объемов данных, обеспечение отказоустойчивости, повышение производительности, максимально дешевый и быстрый перевод продукта в состояние готовности к реальной эксплуатации. Р. О. Дронов отмечает важность, что при разработке, а затем и развертывании продукта необходим мониторинг, балансировка нагрузки, обеспечение быстрого восстановления работоспособности в случае сбоя и надежное хранение данных. Однако, выбор базовых инструментов влияет на все последующие этапы разработки и возможность возникновения ошибок или узких мест производительности [9].

В работе [10] авторы описали облачную систему баз данных NoSQL, конструкция которой значительно снижает расходы и позволяет быстро расширять новые модели, при этом бесперебойно поддерживая их полную функциональность с помощью расширений механизмов хранения.

В работе [11] авторы представили новая система Laser планирования запросов с поддержкой буфера под названием. Предлагаемая система не требует предварительного обучения и может подстраиваться под невидимые рабочие нагрузки на лету посредством постоянных обновлений модели и перераспределения запросов. Сокращение времени выполнения запроса примерно на 80% по сравнению с другими традиционными эвристическими методами с относительно низкими дополнительными накладными расходами, добавленными к критическому пути выполнения запроса.

Физическое проектирование и работа БД

1. *Конфигурация многосерверной базы данных*. Для обеспечения масштабируемости и высокой доступности системы используется многосерверная база данных. Одним из основных преимуществ такой архитектуры является возможность обработки большого количества запросов параллельно, что значительно увеличивает производительность системы. В современных информационных системах многосерверные базы данных активно применяются для повышения отказоустойчивости и обеспечения бесперебойной работы.

Основные стратегии многосерверного хранения данных включают:

• Шардирование (Sharding) – метод, при котором данные разделяются на независимые части (шарды) и распределяются между серверами. Это позволяет равномерно распределить нагрузку и избежать узких мест в системе.

• Репликация (Replication) – процесс создания копий базы данных, которые могут использоваться для балансировки нагрузки и обеспечения отказоустойчивости. В случае сбоя одного из серверов реплика может мгновенно принять на себя нагрузку.

• Балансировка нагрузки – использование специальных механизмов (например, HAProxy или Pgpool-II), которые распределяют поступающие запросы между несколькими серверами базы данных, предотвращая перегрузку одного из узлов.

В данной системе используется подход, комбинирующий репликацию и шардирование для обеспечения оптимальной производительности (рис. 1).

```
ALTER SYSTEM SET wal_level = replica;
ALTER SYSTEM SET max_wal_senders = 10;
ALTER SYSTEM SET hot_standby = on;
```

Рис. 1. – Пример настройки репликации в PostgreSQL

2. *Масштабируемость и репликация данных*

В системе применяется стратегия масштабируемости на основе мастер-слейв репликации. В данном подходе основной сервер (мастер) отвечает за запись новых данных, а реплики (слейвы) предназначены для обработки запросов на чтение. Это позволяет значительно разгрузить основной сервер и обеспечить балансировку нагрузки.

Основные компоненты масштабируемости:

• Логическая репликация – передача изменений с главного сервера на резервные, что позволяет снизить нагрузку на основной узел.

• Автоматическое переключение в случае сбоя – использование инструментов, таких как Patroni, позволяющих автоматически переключать нагрузку на реплику в случае выхода из строя мастера.

• Кэширование данных – применение Redis или Memcached для временного хранения часто запрашиваемых данных, что ускоряет выполнение запросов.

Использование этих технологий позволяет обеспечить отказоустойчивость системы и поддерживать её работоспособность даже в случае возникновения аппаратных сбоев или перегрузки отдельных серверов.

3. *Пример запросов к базе данных*

Запросы к базе данных должны быть оптимизированы для минимизации нагрузки и обеспечения высокой скорости работы (рис. 2).

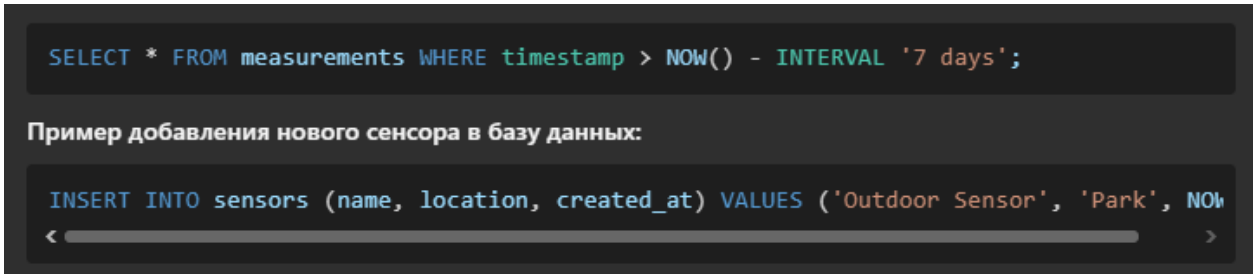

Рис. 2. – Пример настройки репликации в PostgreSQL

Эти запросы демонстрируют основные операции, используемые в системе. Оптимизация запросов и правильная организация индексов позволяют значительно ускорить работу с данными.

Тестирование и анализ работы системы

Для проверки корректности работы системы проводится тестирование с эмуляцией работы множества сенсоров. В процессе тестирования проверяется стабильность системы при увеличении количества сенсоров и нагрузки на базу данных. Имитация работы сенсоров может быть реализована с использованием скриптов на Python, которые отправляют данные в систему в автоматическом режиме. тестовый сценарий позволяет проверить, насколько эффективно система обрабатывает большие объёмы данных в реальном времени (рис. 3).

```python
import requests
import random

API_URL = "http://localhost:8082/measurements"

for _ in range(1000):
    data = {
        "sensor_id": random.randint(1, 50),
        "temperature": random.uniform(-20, 40),
        "humidity": random.uniform(10, 90),
        "pressure": random.randint(980, 1050),
        "rain": random.choice([True, False]),
        "timestamp": "2024-01-26T12:05:00Z"
    }
    requests.post(API_URL, json=data)
```

Рис. 3. – Пример скрипта генерации данных

Для проведения нагрузочного тестирования используются такие инструменты, как Apache JMeter и Locust. Они позволяют эмулировать большое количество одновременных пользователей, отправляющих запросы к системе, и измерять метрики производительности.

Ключевые параметры тестирования:

- среднее время отклика API – сколько времени требуется серверу для обработки запроса;
- пропускная способность БД – максимальное количество обработанных запросов в секунду;
- использование ресурсов сервера – нагрузка на процессор и оперативную память.

На основании результатов тестирования можно определить узкие места в системе и внести необходимые изменения в архитектуру для её оптимизации.

Преимущества реализованной архитектуры:

• Высокая масштабируемость за счёт использования Docker и Apache Kafka.

• Отказоустойчивость благодаря многосерверной базе данных и механизму репликации.

• Гибкость в развертывании, что позволяет быстро вносить изменения в архитектуру системы.

• Упрощённое управление нагрузкой за счёт балансировки запросов между серверами.

Недостатки реализованной архитектуры:

• Сложность конфигурации многосерверной архитектуры, требующая дополнительных знаний и инструментов.

• Повышенные требования к ресурсам, так как использование контейнеров и брокеров сообщений увеличивает нагрузку на сервер.

• Необходимость постоянного мониторинга и управления реплицированными данными.

Тестирование системы показало, что предложенная архитектура является надёжной, масштабируемой и готовой к работе в условиях высоких

нагрузок. Она подходит для использования в приложениях, требующих обработки больших объёмов данных в реальном времени.

Заключение

В результате проведенного исследования получены следующие результаты:

- реализована микросервисная архитектура с использованием REST API;
- оптимизированы базы данных с применением масштабируемости и репликации;
- внедрен распределённый программный брокер Kafka в качестве брокера сообщений для асинхронного взаимодействия сервисов;
- проведена имитация работы множества сенсоров и анализ производительности системы.

В ходе работы также были выявлены некоторые недостатки, связанные со сложностью настройки многосерверной архитектуры и повышенными требованиями к вычислительным ресурсам. Однако предложенные решения позволяют гибко управлять нагрузкой, обеспечивать высокую скорость обработки данных и гарантировать стабильность работы системы.

Таким образом, данное исследование подтвердило эффективность использования многосерверных баз данных и микросервисной архитектуры для обработки большого объёма данных в режиме реального времени. В дальнейшем возможны дополнительные улучшения системы, такие как автоматическое масштабирование контейнеров, улучшение алгоритмов балансировки нагрузки и внедрение более сложных моделей обработки данных.

Работа выполнена при финансовой поддержке Министерства науки и высшего образования Российской Федерации, дополнительное соглашение № 075-02-2025-1538 от 28 февраля 2025 г.

Литература

1. Malleni S.S., Sevilla R., Lema J.C., Bauer A. 2025. Bridging Clusters: A Comparative Look at Multi-Cluster Networking Performance in Kubernetes. In Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE '25). New York. Association for Computing Machinery. 2025. pp. 113–123.

2. Pérez de Prado R., García-Galán S., Muñoz-Expósito J.E., Marchewka A., Ruiz-Reyes N. Smart Containers Schedulers for Microservices Provision in Cloud-Fog-IoT Networks. Challenges and Opportunities. Sensors 2020. 20. 1714.

3. Leang B., Ean S., Ryu G.-A., Yoo K.-H. Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment. Sensors. 2019. 19. 134.

4. Olivier M., Stefan S., Habib M. L3: Latency-aware Load Balancing in Multi-Cluster Service Mesh. In Proceedings of the 25th International Middleware Conference (Middleware '24). New York. Association for Computing Machinery. 2024. pp. 49–61.

5. Карпович М. Н. Особенности проектирования микросервисно-событийных архитектур для высоконагруженных распределенных систем обработки информаций // Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2023. № 1(266). С. 89-95.

6. Бондаренко А. С., Королев Д. В., Зайцев К. С. Исследование эффективности использования мультиагентных моделей в современных микросервисных архитектурах // International Journal of Open Information Technologies. 2024. Т. 12, № 8. С. 48-55.

7. Щеглов Г. А., Жданова К. А., Жумаев З. С., Каменев Н. Д. Архитектура открытого программного комплекса UEMKA для управления целевыми устройствами SMART-наноспутников // Труды Института системного программирования РАН. 2024. Т. 36, № 5. С. 163-180.

8. Polyantseva K., Gorodnichev M., Moseva M. Ensuring the Reliability of a Highly Loaded Vehicle Monitoring and Traffic Control Platform // Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow. 2023. pp. 1-8.

9. Дронов Р. О. Особенности выбора архитектуры и компонентов для построения распределенных приложений // Информатика: проблемы, методы, технологии : Материалы XX Международной научно-методической конференции, Воронеж, 13–14 февраля 2020 года. 2020. С. 1246-1250.

10. Lei H., Li C., Zhou R., Zhu J., Yan K., Xiao F., Xie M., Wang J., Di S. X-Stor: A Cloud-Native NoSQL Database Service with Multi-Model Support. Proc. VLDB Endow. 17, 12 (August 2024), 2024. pp. 4025–4037.

11. Yuwei Huang Y., Li G. Laser: Buffer-Aware Learned Query Scheduling in Master-Standby Databases. Proc. VLDB Endow. 18, 3 (November 2024), 2024. pp. 743–755.

References

1. Malleni S.S., Sevilla R., Lema J.C., Bauer A. 2025. Bridging Clusters: A Comparative Look at Multi-Cluster Networking Performance in Kubernetes. In Proceedings of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE '25). New York. Association for Computing Machinery. 2025. pp. 113–123.

2. Pérez de Prado R., García-Galán S., Muñoz-Expósito J.E., Marchewka A., Ruiz-Reyes N. Smart Containers Schedulers for Microservices Provision in Cloud-Fog-IoT Networks. Challenges and Opportunities. Sensors 2020. 20. 1714.

3. Leang B., Ean S., Ryu G.-A., Yoo K.-H. Improvement of Kafka Streaming Using Partition and Multi-Threading in Big Data Environment. Sensors. 2019. 19. 134.

4. Olivier M., Stefan S., Habib M. L3: Latency-aware Load Balancing in Multi-Cluster Service Mesh. In Proceedings of the 25th International Middleware Conference (Middleware '24). New York. Association for Computing Machinery. 2024. pp. 49–61.

5. Karpovich M. N. Trudy BGTU. Seriya 3: Fiziko-matematicheskiye nauki i informatika. 2023. № 1(266). pp. 89-95.

6.Bondarenko A. S., Korolev D. V., Zaytsev K. S. International Journal of Open Information Technologies. 2024. V. 12, № 8. pp. 48-55.

7.Shcheglov G. A., Zhdanova K. A., Zhumayev Z. S., Kamenev N. D. Trudy Instituta sistemnogo programmirovaniya RAN. 2024. V. 36, № 5. pp. 163-180.

8. Polyantseva K., Gorodnichev M., Moseva M. Systems of Signals Generating and Processing in the Field of on Board Communications, Moscow, Russian Federation, 2023. pp. 1-8.

9. Dronov R. O. Informatika: problemy, metody, tekhnologii : Materialy XX Mezhdunarodnoy nauchno-metodicheskoy konferentsii, Voronezh, 13–14 fevralya 2020 goda. 2020. pp. 1246-1250.

10. Lei H., Li C., Zhou R., Zhu J., Yan K., Xiao F., Xie M., Wang J., Di S. X-Stor: A Cloud-Native NoSQL Database Service with Multi-Model Support. Proc. VLDB Endow. 17, 12 (August 2024). 2024. pp. 4025–4037.

11. Yuwei Huang Y., Li G. Laser: Buffer-Aware Learned Query Scheduling in Master-Standby Databases. Proc. VLDB Endow. 18, 3 (November 2024). 2024. pp. 743–755.

Дата поступления: 12.05.2025

Дата публикации: 25.06.2025