**УДК 004.62**

**ЭФФЕКТИВНОСТЬ ИСПОЛЬЗОВАНИЯ ИНСТРУМЕНТОВ АГРЕГАЦИИ ДАННЫХ ПРИ РАБОТЕ С ГЕТЕРОГЕННЫМИ ИСТОЧНИКАМИ**

**ТАБЛИЧНЫЙ ФОРМАТ ДЛЯ ОЗЕРА ДАННЫХ ICEBERG**

Быченков А.К., Матчин В.Т.

Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА – Российский технологический университет», Москва

1. Использование ETL-инструмента для выгрузки данных в разные форматы; Apache NiFi.
2. Микширование данных в Dremio/Presto/Trino – SQL Query Engine;
3. Репликация в Clickhouse (PostgreSQL, MySQL). Коннекторы, драйвера;
4. PostgreSQL - репликация, Tarantool – ETL.

PostgreSQL, NiFi, Data Lakehouse (Iceberg: Metadata for Parquet files, Hudi), Dremio/Presto/Trino

(S3 CSV – ) PostgreSQL -

Преимущества:

- актуальность данных

Data movement, distributed, aggregation, orchestration, workflows,

Есть два подхода: прямое обращение к источникам данных DirectQuery, выгрузка данных.

ODBC

JDBC

Форматы хранения Data Lakehouse (Iceberg, Delta Lake, Hudi)

В эпоху цифровой трансформации и массового накопления данных сбор, хранение и анализ информации становятся ключевыми аспектами успешной деятельности организаций во всех сферах. С каждым днем возрастает сложность этой задачи в связи с ростом объемов, разнообразием и скоростью поступления данных из различных источников. В такой среде инструменты агрегации данных, такие как Data Lake, Data Warehouse и более новый концепт Data Lakehouse, выступают в роли стратегических ресурсов, обеспечивая не только хранение, но и эффективную обработку, анализ и использование информации. В данной статье мы рассмотрим эффективность применения этих инструментов при работе с гетерогенными источниками данных, а также выявим основные преимущества и вызовы, с которыми сталкиваются организации в процессе их использования.

1. Микширование данных в Dremio/Presto/Trino – SQL Query Engine;

Data Lake (набор файлов (Parquet, AVRO, ORC), хранятся в S3, HDFS) Storage +

Data Lakehouse (Iceberg, Hudi, Delta Lake) Structured Storage

Warehouse (Clickhouse) -

В современном информационном ландшафте данные являются не только ключевым активом, но и центральным фундаментом для принятия стратегически важных решений. С развитием технологий и появлением новых методов обработки данных, организации сталкиваются с необходимостью эффективного управления гетерогенными источниками данных. В этой статье будет исследована эффективность использования инструментов агрегации данных в контексте Data Lake, Data Warehouse и Data Lakehouse, сосредотачиваясь на их способности обеспечить согласованное хранение, обработку и анализ данных из разнообразных источников.

В сфере баз данных и управления информацией производительность различных систем играет ключевую роль в обеспечении эффективности бизнес-процессов. Одной из важнейших характеристик является способность обрабатывать параллельные транзакции с высокой производительностью. В этой статье будет проведен сравнительный анализ производительности ACID-совместимых СУБД (систем управления базами данных), Tarantool и PostgreSQL, при обработке параллельных транзакций. Такое сопоставление может помочь в выборе наиболее подходящей системы, способной обрабатывать большее количество транзакций при приемлемом уровне задержки, сохраняя согласованность данных.

ACID (англ. Atomicity, Consistency, Isolation, Durability) — это базовые принципы, обеспечивающие надежность транзакционных операций в базах данных [1]. Для обеспечения изоляции транзакций рассматриваемые СУБД, Tarantool и PostgreSQL, предоставляют наивысший уровень изоляции – Serializable.

Для СУБД Tarantool уровень изоляции Serializable является стандартным, что обеспечивает отсутствие известных аномалий изоляции при выполнении транзакций (грязное чтение, неповторяемое чтение, фантомное чтение). Данный уровень изоляции является следствием того, что Tarantool обрабатывает транзакции в одном потоке [2], поэтому все транзакции выполняются строго последовательно. Именно эта особенность позволяет обеспечить наивысший уровень изоляции, а также избежать блокировок.

СУБД PostgreSQL по умолчанию предоставляет уровень изоляции Read Committed, который при необходимости можно изменить на Serializable. Уровень изоляции Serializable в PostgreSQL основан на пессимистическом контроле параллелизма [3]. Он гарантирует согласованность, предполагая, что две транзакции могут пытаться обновить одни и те же данные, и использует предикатные блокировки, чтобы гарантировать, что они этого не сделают.

Для сравнения рассматриваемых СУБД было проведено нагрузочное тестирование в контексте обработки множества одновременных банковских транзакций. В каждой СУБД была создана таблица (в контексте СУБД Tarantool аналогом таблицы является схема) для хранения счетов пользователей. Каждая строка содержит информацию по двум столбцам: уникальный идентификатор счета и баланс.

Условия проведения тестирования:

СУБД работает на одном узле без шардирования и репликации;

данные записываются на диск (используется дисковый движок vinyl в Tarantool [4]);

1000 одновременных пользователей;

обеспечение согласованности данных с использованием уровня изоляции Serializable.

Для взаимодействия с рассматриваемыми СУБД был использован язык Python вместе с веб-фреймворком FastAPI. Для работы с пользовательскими счетами был создан HTTP API (интерфейс взаимодействия, работающий по протоколу HTTP – протокол передачи гипертекста). Для упрощения тестирования не используется механизм аутентификации.

Пользователям предоставляется следующий HTTP API:

создание счета (возвращает уникальный идентификатор);

просмотр счета по уникальному идентификатору (возвращает баланс счета);

просмотр всех счетов (используется для нахождения другого счета для осуществления перевода);

внос суммы на счет (прибавление определенной суммы к балансу счета);

снятие суммы со счета;

перевод денежных средств на другой счет;

удаление счета (при этом все средства, находящиеся на удаляемом счету, переводятся на случайный счет);

просмотр общего баланса системы (используется для проверки согласованности данных);

обнуление балансов всех счетов.

В качестве инструмента нагрузочного тестирования HTTP API был выбран Python-фреймворк Locust. В ходе тестирования каждую секунду создается 10 виртуальных пользователей, которые начинают работать с системой. Число пользователей увеличивается вплоть до 1000.

Перед первой операцией каждый пользователь отсылает запрос на создание счета. Затем раз в 1-5 секунд отправляет либо запрос на пополнение денег на счете (вероятность этого события равна 8.33%), либо на снятие (8.33%), либо на перевод (75%). Также пользователь может удалить счет, затем сразу создать новый (8.33%).

Главным критерием проверки согласованности данных в системе является общая сумма денежных средств на всех счетах. Во время взаимодействия с системой пользователи могут вводить и выводить денежные средства только по 100 единиц. Переводы могут осуществляться на любую сумму больше нуля с точностью до 2 цифр после запятой. Таким образом баланс денежных средств в системе в каждый момент времени должен быть кратен 100. Этот баланс может быть легко нарушен при стандартном уровне изоляции PostgreSQL Read Committed, вследствие возникновения аномалий неповторяющегося и фантомного чтений [5]. Например, если в одной транзакции осуществляется перевод денежных средств, а в другой снятие денег с этого же счета или удаление счета получателя. Тогда сумма перевода может исчезнуть из системы, что отразится на общем балансе.

Для проверки согласованности данных используется еще одна группа виртуальных пользователей, которая постепенно увеличивается с 1 до 100. Каждую секунду они отправляют запросы на случайные переводы между счетами и проверяют кратность общего баланса системы.

Описанное выше нагрузочное тестирование для СУБД PostgreSQL и Tarantool было проведено. Проанализированы значения RPS (количество обработанных запросов в секунду) и медианное время отклика, полученные на одном узле с процессором AMD Ryzen 5500U (12 потоков), 16 GB оперативной памяти и твердотельным накопителем NVMe M.2 512 GB. Результаты приведены в таблице 1, содержащей следующие столбцы:

СУБД – наименование СУБД;

max RPS – максимальное полученное количество обработанных запросов в секунду;

кол-во пользователей при max RPS – количество виртуальных пользователей в момент достижения максимального полученного количества обработанных запросов в секунду;

RPS при 1000 пользователей – количество обработанных запросов в секунду в момент достижения нагрузки в 1000 виртуальных пользователей;

50% response time > 100 ms (кол-во пользователей) – количество пользователей, начиная с которого медианное время отклика начинает превышать 100 миллисекунд;

50% response time при 1000 пользователей – медианное время отклика в момент достижения нагрузки в 1000 виртуальных пользователей.

Таблица 1 – Результаты нагрузочного тестирования

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СУБД | Max RPS | Кол-во пользователей при max RPS | RPS при 1000 пользователей | 50% response time > 100 ms  (кол-во пользователей) | 50% response time при 1000 пользователей |
| Tarantool | 615 RPS | 680 | 475 RPS | 590 | 550 ms |
| PostgreSQL | 349 RPS | 450 | 219 RPS | 420 | 640 ms |

На рисунке 1 показан график динамики RPS для Tarantool. При достижении 1000 пользователей значение RPS составило 475.2, затем держалось между 390-490. Максимальное достигнутое значение RPS – 615.



Рисунок 1 – Нагрузочное тестирование СУБД Tarantool

На рисунке 2 показан график динамики RPS для PostgreSQL. При достижении 1000 пользователей значение RPS составило 219.5, затем держится между 180-240. Достигает максимальной производительности в 349 RPS.

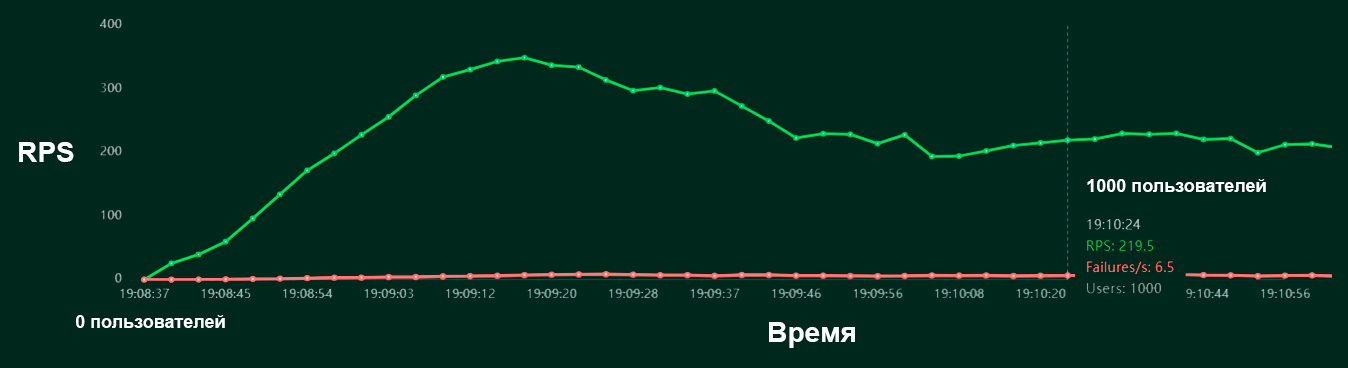


Рисунок 2 – Нагрузочное тестирование СУБД PostgreSQL

Исходя из полученных результатов, можно сделать вывод: СУБД Tarantool в сопоставимых условиях предоставляет лучшую производительность, позволяя обрабатывать более чем в 1.5 раза большее количество транзакций в секунду.

В заключение, хотя результаты сравнительного анализа производительности между СУБД Tarantool и PostgreSQL и указывают на превосходство Tarantool в условиях высокой нагрузки, при принятии решения о выборе конкретной СУБД следует учитывать не только производительность, но и другие факторы, такие как требования к функциональности, масштабируемость, стоимость внедрения и поддержки.

**Список использованных источников:**

1. Кузнецов, С. Д. Транзакционные параллельные СУБД: новая волна / С. Д. Кузнецов // Труды Института системного программирования РАН. – 2011. – Т. 20. – С. 189-251.

2. Tarantool transaction model [Электронный ресурс]. – URL: <https://www.tarantool.io/en/doc/latest/concepts/atomic/transaction_model/> (дата обращения 26.03.2024).

3. Serializable vs. Snapshot Isolation Level [Электронный ресурс]. – URL: <https://habr.com/ru/articles/662566/> (дата обращения 26.03.2024).

4. Аникин, Д. Tarantool: СУБД с хранением в памяти и сервер приложений / Д. Аникин, С. Пугачев // Открытые системы. СУБД. – 2017. – № 2. – С. 32-34.

5. Тетенькин, А. Ю. Проблемы параллельного выполнения транзакций в базах данных / А. Ю. Тетенькин // Татищевские чтения: актуальные проблемы науки и практики : материалы XVII Международной научно-практической конференции: в 3 томах, Тольятти, 24–25 апреля 2020 года. Том 3. – Тольятти: Волжский университет имени В.Н. Татищева (институт), 2020. – С. 16-19.

**© Быченков А.К., Матчин В.Т., 2024**