УДК 004.65; 004.75

Методология бесшовной миграции нагруженных СУБД на примере MongoDB и YDB

Хайрнасов А.К.
Научный руководитель – старший преподаватель фПИН, Береснев А.Д.
Университет ИТМО
312343@niuitmo.ru

Введение

Сегодня в распределенных серверных приложениях выбор СУБД сильно влияет на устойчивость к сбоям, на возможность роста, на стоимость работы и на развитие сервисов. Рост данных и рост требований к доступности заставляют команды использовать нереляционные СУБД, такие как MongoDB. Но в то же время появился новый класс распределенных транзакционных СУБД, которые умеют горизонтально расти и при этом дают строгие гарантии согласованности. К этим СУБД относится YDB.

MongoDB удобно использовать, когда нужно быстро собрать прототип или запустить сервис с большой нагрузкой [1]. У MongoDB гибкая схема, но при росте нагрузки и усложнении логики уже появляются ограничения. У MongoDB слабая модель консистентности, схема может быть слишком свободной, шардировать вручную несет риски стабильности сервиса, а репликацию и балансировку нужно поддерживать отдельно. YDB решает эти проблемы [2]. YDB автоматически делит данные на шарды, работает предсказуемо под нагрузкой и поддерживает полные ACID-транзакции. Поэтому YDB выглядит хорошей альтернативой, если важна строгая консистентность и хочется сократить расходы. В работе сравниваются стратегии миграции, так как за время жизни сервиса накопилось несколько десятков терабайт сырых данных, которые надо перенести без простоя сервиса для пользователя.

Цель работы − сравнить механизмы миграции, учитывая потребность в минимальном простое сервиса, переносе данных без потерь.

Основная часть

Модель данных этих СУБД отличается, так что нужно с осторожностью построить индексы и таблицы YDB для эффективного взаимодействия. Учитывая роль базы как архивной хранилище, вполне подходит схема с вынесением ключевых полей необходимых для сортировки в отдельные столбцы, по которым можно создать индексы, а основное тело данных положить в один столбец типа JSON для более простого расширения полей сущности.

Проведено сравнение готовых стратегий миграции данных: перенос всех данных за раз с отключением сервиса, что не подходит из-за недоступности системы для пользователей; Change Data Capture, когда происходит чтение лога изменений, но с данным подходом есть риски деградации базы источника или потери данных при ошибках миграции; перенос исторических данных через пользовательское приложение, что дает более гибкие настройки и возможности следить за процессом; двойная запись в две системы, что хорошо подходит для архивных систем, где основная нагрузка нацелена на идемпотентную запись данных [3].

В итоге учитывая специфику системы хорошо подходит двойная запись для новых данных, а старый данные переносить фоновым процессом на уровне приложения. Основной ключ представляет из себя хэш-подобную строку, следовательно пространство значений основного ключа можно разбить на разные отрезки, что позволяет проводить миграцию старых значений параллельно на разных инстансах приложения. С таким подходом можно настраивать скорость переноса исторических данных исходя из нагрузки на систему и базы данных.

Для валидации перенесенных данных полезен подход теневых чтений: на пользовательский запрос выдаются данные из старой СУБД, но в фоне аналогичный запрос идет в новую базу и сравнивается результат. При расхождениях пишутся логи, по которым можно понять, при каком запросе и какие сущности не сошлись между выдачами двух СУБД.

После завершения валидации данных следует шаг постепенного переключение трафика на новую СУБД. Для этого выбрано поле, которое участвует во всех запросах фильтрации, от него берется хэш и идет проверка, попадает ли он в процентную группу для переключения на новый источник данных [4].

Выводы

В результате данной работы была построена методология бесшовной миграции в нагруженном приложении с большими объемами данных без недоступности системы для пользователей.

Из рассмотренных стратегий для архивного профиля нагрузки наиболее хорошо подходит двойная запись и фоновая миграция архивных данных. Данный подход позволяет гибко настраивать нагрузку на базу и имеет относительно небольшие риски деградации сервиса. Процесс можно горизонтально масштабировать, так как пространство ключей разбивается на отрезки.

У данного подхода есть ограничение в виде того, что двойная запись требует идемпотентность операций. Если требуется более сложная логика, то нужно как-то делать транзакции в двух системах, либо использовать подход CDC и аккуратно применять изменения в новую базу, минимизируя риски потерь данных и деградации базы источника.

Литература

1. Статья «MongoDB and Data Consistency: Bridging the Gap between Performance and Reliability». DOI: 10.32996/jcsts.2024.6.2.21 (дата обращения 07.02.2026).

2. Статья «A Unified, Practical, and Understandable Model of Non-transactional Consistency Levels in Distributed Replication». DOI: 10.48550/arXiv.2409.01576 (дата обращения 07.02.2026).

3. Статья «Designing a Zero Downtime Migration Solution with Strong Data Consistency – Part IV». URL: https://engineering.mercari.com/en/blog/entry/20241113-designing-a-zero-downtime-migration-solution-with-strong-data-consistency-part-iv (дата обращения 07.02.2026).

4. Статья «Strangler Fig pattern». URL: https://learn.microsoft.com/en-us/azure/architecture/patterns/strangler-fig (дата обращения 07.02.2026).