САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Безопасность систем баз данных

Лекция 7
Репликация.
Балансировка нагрузки

Агафонов Антон Александрович
к.т.н., доцент кафедры ГИиИБ

Самара

План лекции

- Понятие репликации
- Типы репликации
- Виды топологии репликации
- Балансировка нагрузки
- Репликация:
  - в MySQL
  - в PostgreSQL
  - в MongoDB

Репликация

Репликация - набор технологий копирования и распространения данных и объектов баз данных между базами данных и последующей синхронизации баз данных для поддержания их согласованности.

Задачи, решаемые репликацией:

- *Повышение производительности системы*. Создание реплик позволяет распределить запросы между серверами, тем самым снизив нагрузку на каждый отдельный сервер.
- *Обеспечение отказоустойчивости*. В случае отказа или необходимости обслуживания одного из серверов можно быстро восстановить работоспособность системы, переведя его нагрузку на одну из реплик.
- *Незаметное резервирование данных*. При наличии реплики выполнять резервное копирование можно на ней – это позволит резервировать данные без снижения производительности всей системы.
- *Выделение серверов аналитики*. Ресурсоемкие задачи анализа данных можно выполнять на выделенной реплике без влияния на производительность всей системы.
- *Снижение транспортных издержек*. В распределенных системах значительный вклад в общее время обработки запросов вносит время передачи данных по сети. Уменьшить это время можно, направляя запросы клиентов на наименее удаленные реплики.

Типы репликации

- **Синхронная репликация.** При таком типе репликации транзакция, модифицирующая данные, считается подтверждённой только тогда, когда все серверы подтвердят эту транзакцию. При синхронной репликации гарантировано то, что при отказе ведущего сервера не произойдёт потери данных, а также то, что все серверы возвращают согласованные данные вне зависимости от того, к какому серверу был запрос. Синхронная репликация — это максимально надежный вариант, но и максимально медленный.
- **Асинхронная репликация.** При таком типе репликации транзакция, модифицирующая данные, считается подтверждённой тогда, когда она подтверждена локально на том сервере, на котором эти изменения происходили. При асинхронной репликации высока вероятность потери данных при отказе ведущего сервера, а также нет никаких гарантий согласованности данных в произвольный момент времени, т.к. изменения могут дойти до других серверов с большой задержкой, или не дойти вовсе. Асинхронная репликация — это максимально быстрый вариант, но и максимально ненадежный.

Типы репликации

- **Полусинхронная репликация.** При таком типе репликации транзакция, модифицирующая данные, считается подтверждённой тогда, когда хотя бы один из других серверов подтвердит получение (но не применение!) изменений. Такой тип репликации является неким компромиссным решением. При полусинхронной репликации вероятность потери данных при отказе ведущего сервера крайне мала (только если откажет и сервер, подтвердивший получение изменений), а вот гарантии согласованности данных в произвольный момент времени также нет, т.к. изменения могут быть применены с задержкой. Полусинхронная репликация — это более быстрый тип репликации, чем синхронная и более надежный, чем асинхронная.

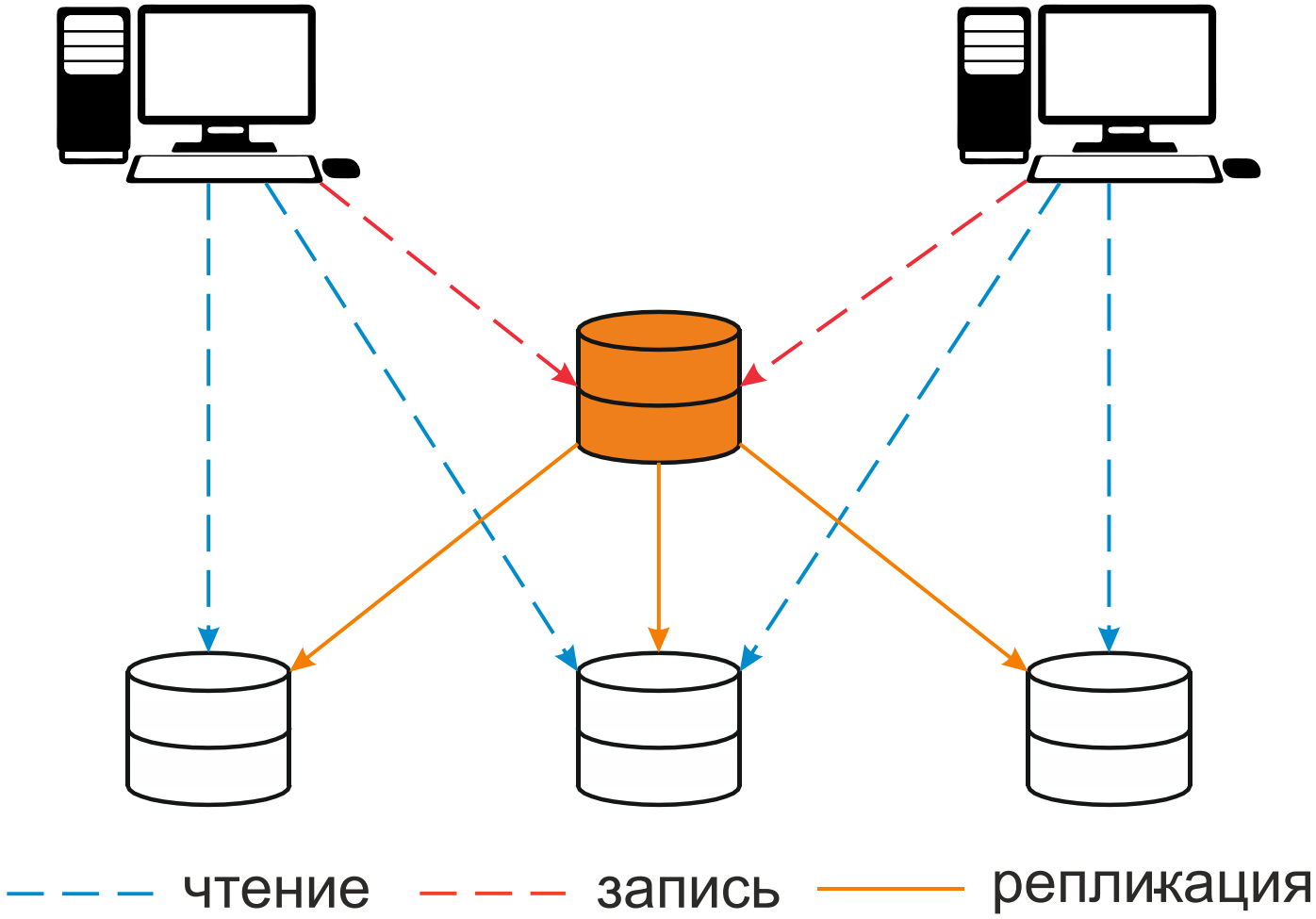
Виды топологии репликации

В контексте репликации у серверов могут быть две роли: **ведущий** сервер и **подчиненный** (**ведомый**) сервер. При репликации данные копируются с ведущего сервера на подчиненные серверы.

Количество серверов с этими ролями определяет различные виды топологии репликации:

- *Репликация с одним ведущий сервером*. Данные всегда отправляются на один конкретный ведущий узел.
- *Репликация с несколькими ведущими серверами*. Может существовать несколько узлов, играющих роль ведущих, и каждый ведущий узел должен сохранять данные в кластере.
- *Репликация без ведущих серверов*. Ожидается, что все узлы могут принимать данные при записи.

Репликация с одним ведущим сервером



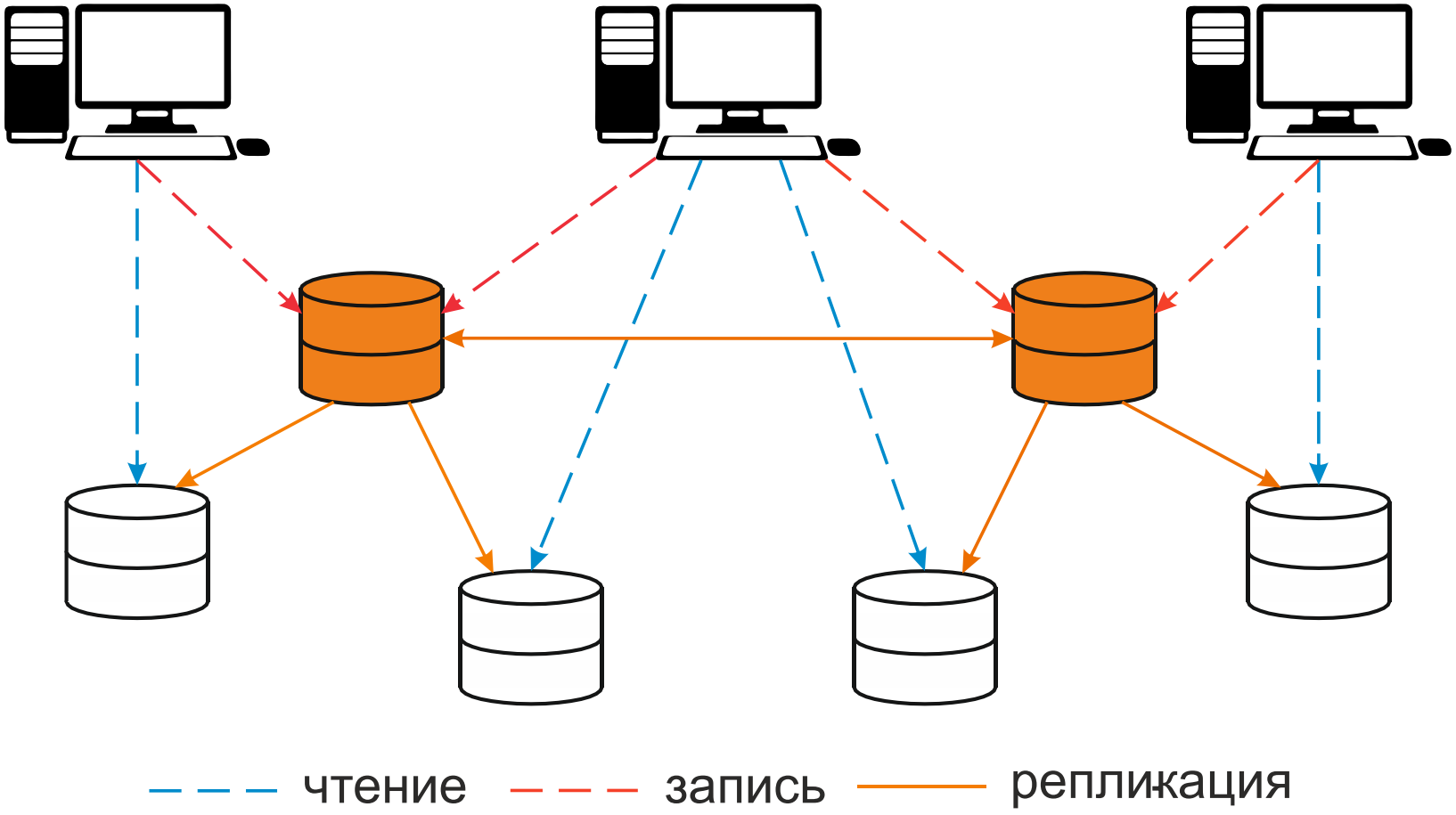
Репликация с одним ведущим сервером

Метод репликации с одним ведущим узлом очень популярен благодаря своей простоте и позволяет гарантировать следующее:

- ▲ Отсутствие конфликтов согласованности в данных, т.к. все операции записи исходят из одного узла.
- ▲ Если предположить, что все операции детерминированные, они приведут к одинаковым результатам на каждом узле.

- ▼ В такой топологии может возникнуть проблема, что один ведущий сервер не способен обработать все запросы записи. Однако большинство приложений выполняет значительно больше запросов на чтение, чем на запись, поэтому возникновение такой ситуации маловероятно.
- ▼ В случае значительно удаленных серверов возможна ситуация, когда чтение будет происходить быстро – из локальной копии данных на ближайшем ведомом сервере, а запись с задержкой – на удаленном ведущем сервере.

Репликация с несколькими ведущими серверами



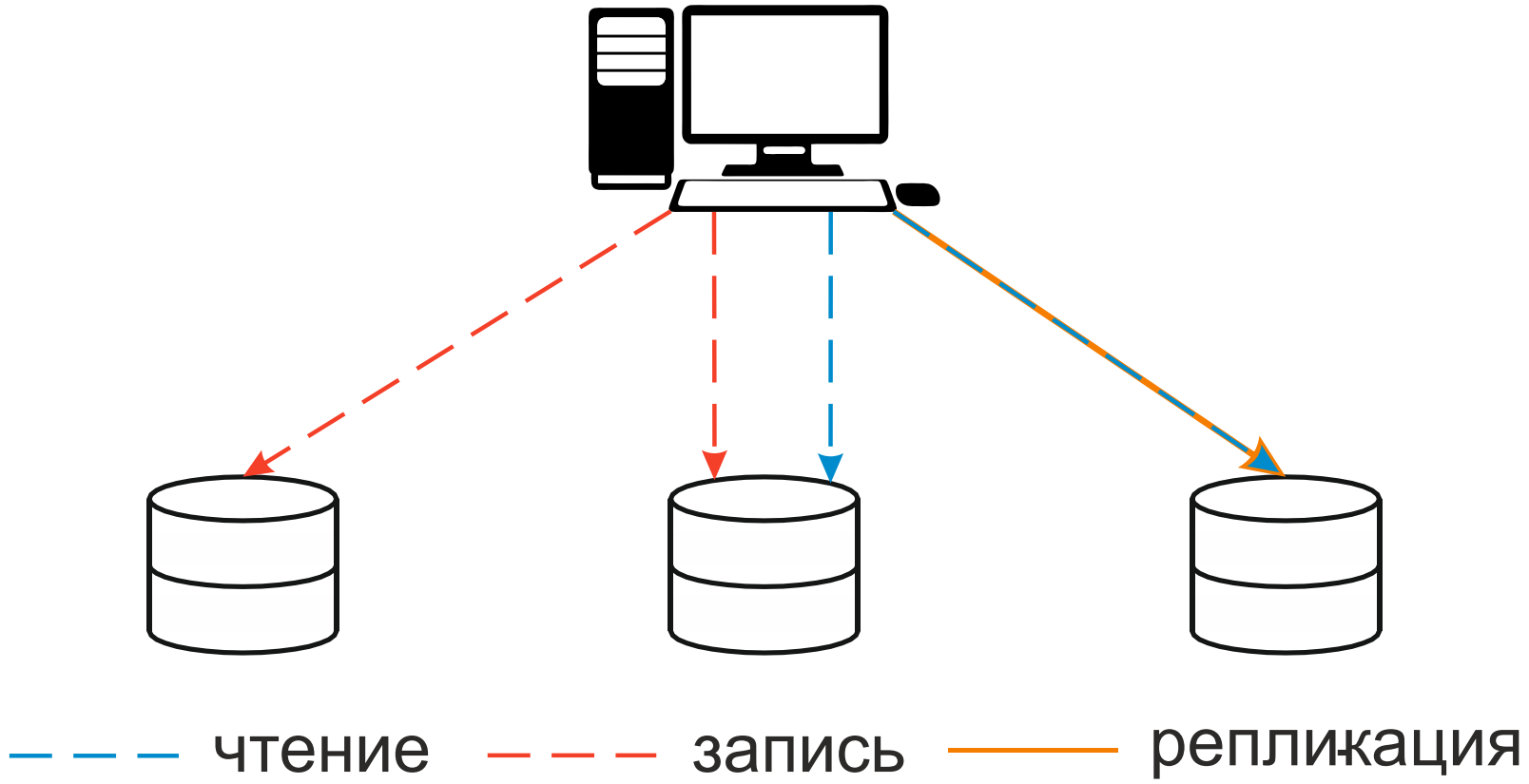
Репликация с несколькими ведущими серверами

Основная проблема топологии заключается в возникновении *конфликтов изменения данных* и необходимости их разрешения. Стоит отметить, что конфликты в такой топологии могут возникать только при условии асинхронной репликации.
Теоретически, имея несколько ведущих серверов синхронной репликации, можно избежать конфликтов, но в этом случае фактически утрачивается масштабируемость запросов на запись – то, ради чего собственно данная топология и рассматривалась.

Если структура данных позволяет разделить их на непересекающиеся множества и гарантировать, что каждый ведущий сервер работает на запись только со своей частью данных, а остальные данные использует на чтение, то возникновения конфликтов можно избежать. В этом случае исключается возможность возникновения конфликтов, и все операции записи обрабатываются на ближайшем сервере. В случае, если информационная система не позволяет разделить свои данные подобным образом, приходится применять механизмы разрешения конфликтов:

- использование стратегии, в которой каждой операции записи присваиваются временные метки, и затем всегда применяется последняя запись (стратегия Last Write Wins);
- использование пользовательского кода разрешения конфликтов и т.д.

Репликация без ведущих серверов



Репликация без ведущих серверов

Основная идея: клиент посылает запрос на запись одновременно нескольким репликам, и как только он получает подтверждение от некоторых из них (в предельном случае от одной), считается, что запись прошла успешно, и клиент может продолжать работу.

Проблема: допустим, запрос на запись будет завершен удачно на двух репликах из трех, а на одной не завершится. В этом случае будет две реплики с новым значением и одна со старым, и запросы чтения к разным репликам будут возвращать разные значения.

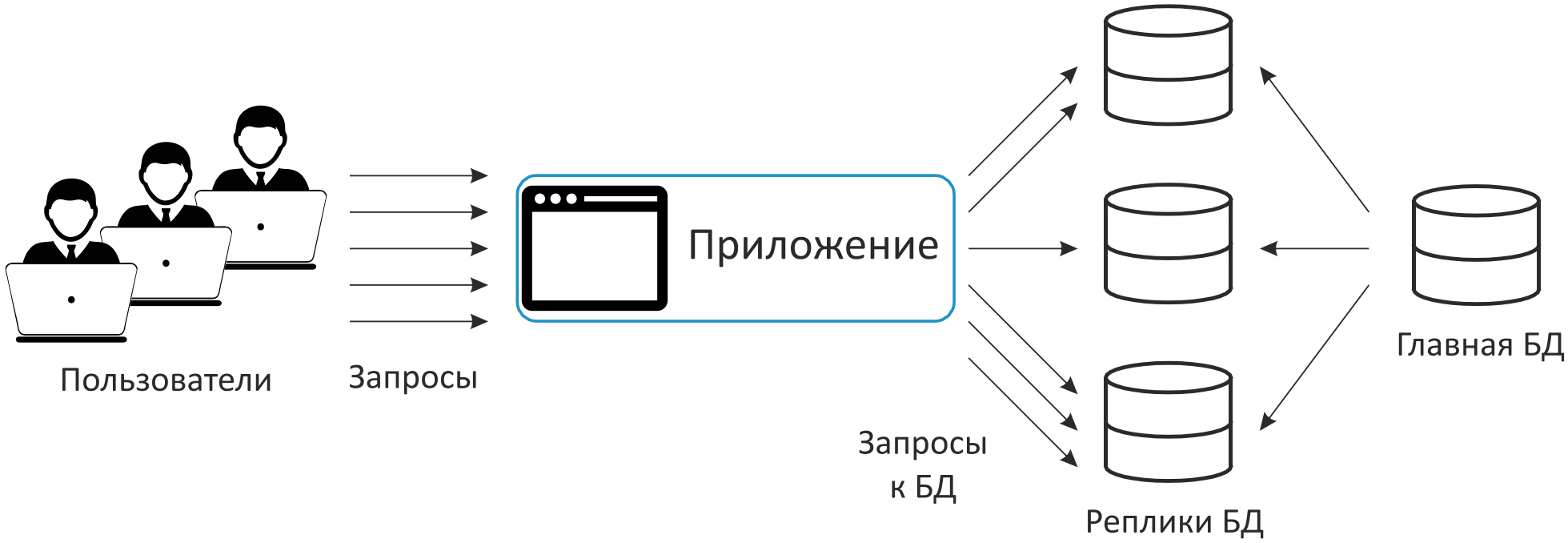
В отсутствии ведущего сервера, обеспечивающего синхронизацию, проблема рассогласования данных решается следующим образом:

1) Клиент посылает запросы на чтение также на несколько реплик одновременно.
2) Реплики возвращают свои локальные значения и некоторый номер версии данных, который используется клиентом, чтобы решить, какое из полученных значений является актуальным.
3) В это же время выполняется и синхронизация значений между репликами: если при чтении данных обнаруживается (по номеру версии), что на некотором узле значение не актуально, клиент посылает этому узлу запрос на запись с актуальным значением. Такой механизм называют чтением с восстановлением (Read Repair).

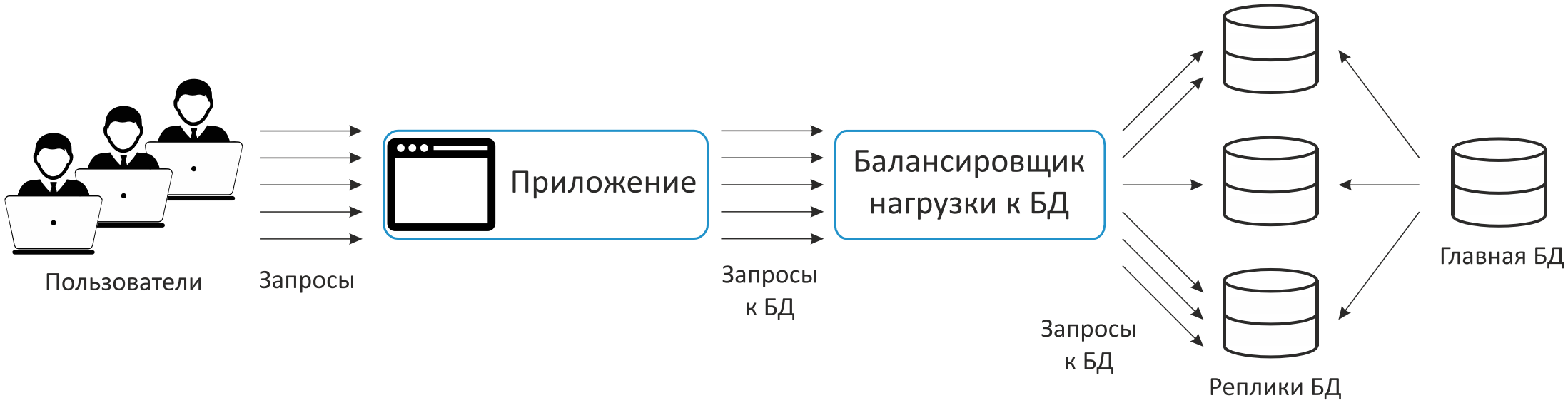
Балансировка нагрузки

В ситуациях, когда одна база данных не справляется с нагрузкой, можно завести несколько баз, настроить репликацию в них главной базы и организовать приложение так, чтобы оно направляло разные запросы разным базам.

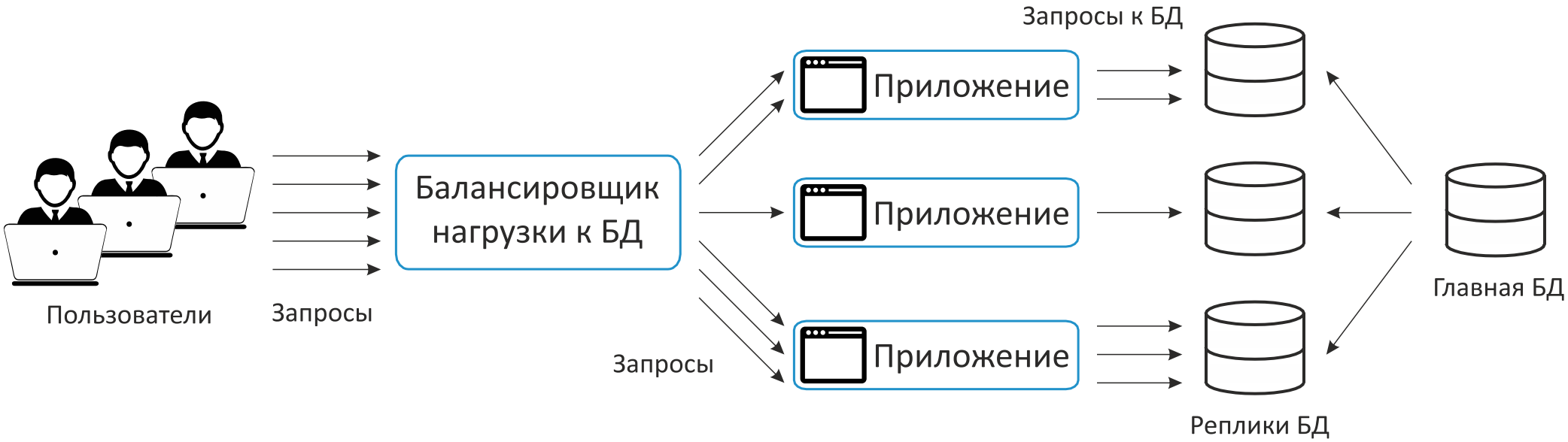
Приложение должно запрашивать данные у разных баз, но для этого необходима специальная архитектура уровня доступа к данным.



Другой вариант – использование балансировщика нагрузки между БД: MySQL Proxy для MySQL, Pgpool-II для PostgreSQL и т.д. Такие программы обычно предоставляют SQL-интерфейс, так что приложения могут работать так, будто это настоящий сервер БД. Балансировщик в свою очередь направляет запросы к базам, у которых в данный момент меньше всего запросов.



Еще одна возможность – масштабировать приложение вместе с базами данных, так чтобы каждый экземпляр приложения подключался к своему экземпляру базы данных. В таком случае пользователь должен подключаться к одному из нескольких экземпляров приложения, что легко реализовать с помощью балансировщика HTTP-нагрузки.



Механизмы репликации в MySQL

В модели репликации MySQL выделяют два ключевых физических уровневых компонента:

- ведущий сервер (source / master) – сервер, предоставляющий данные;
- подчиненный сервер (replica / slave) – сервер, копирующий себе данные, предоставляемые ведущим сервером.

Фактически в MySQL реализован один тип репликации, основанный на распространении данных двоичных журналов.

MySQL. Репликация двоичных журналов

Этот тип репликации основан на том, что все изменения данных, происходящие на мастере, после первоначальной синхронизации мастера и реплик повторяются на репликах.

Запросы на изменение данных выполняются только на мастере, а запросы на чтение данных могут выполняться как на репликах, так и на мастере. Процесс репликации на одной реплике не влияет на работу других реплик, и практически не влияет на работу мастера.

- Первоначальная синхронизация данных выполняется с помощью резервной копии данных мастера. Репликация данных производится при помощи двоичного журнала, ведущегося на мастере.
- На каждом подчиненном сервере есть канал репликации, который служит для передачи информации об изменениях данных от мастера к реплике. С каждым каналом репликации ассоциирован приемник данных (поток ввода/вывода), один или несколько исполнителей команд (поток исполнения SQL) и журнал репликации (Relay Log).
- С помощью приемника данных информация об изменениях из двоичного журнала на мастере копируется в журнал репликации на подчиненном сервере.
- С помощью исполнителя команд события в журнале репликации воспроизводятся на данных реплики, тем самым происходит согласование данных мастера и реплик.

MySQL. Репликация двоичных журналов

При репликации передаются не сами измененные данные, а только информация, необходимая для воспроизведения этих изменений. В настоящее время в СУБД MySQL поддерживаются три формата ведения двоичного журнала:

- Протоколирование выражений (Statement-based Logging). События в журнале представляют собой SQL-выражения, которые были выполнены при модификации данных.
- Протоколирование записей (Row-based Logging). События в журнале представляют собой изменения в отдельных записях таблицы данных в двоичном формате.
- Смешанное протоколирование (Mixed Logging). Часть событий представлено в виде SQL-операторов, часть - в виде изменений отдельных записей.

MySQL. Репликация двоичных журналов

Протоколирование выражений:

- ▲ Как правило, более компактно, особенно когда речь идет о массовых изменениях данных.
- ▲ Поскольку файлы журнала содержат все выражения, изменявшие данные, такой журнал может использоваться для аудита БД.
- ▼ При воспроизведении изменений придется повторять вычисление ресурсоемких запросов на реплике, что не лучшим образом повлияет на скорость синхронизации.
- ▼ SQL-выражения могут содержать ряд недетерминированных функций, например, `NOW()`, `RAND()`, `USER()` и т.д. Исполнение этих выражений на мастере и на реплике может привести к разным результатам и, соответственно, к рассогласованию данных.

Протоколирование записей

- ▲ Режим максимально безопасен с точки зрения согласованности данных, т.к. реплицируются непосредственно измененные строки данных, и они одинаковы независимо от сервера, на котором эти изменения воспроизводятся.
- ▼ Использование этого режима приводит к быстрому росту объема двоичного журнала.

В режиме смешанного протоколирования по умолчанию используется протоколирование выражений, но в определенных случаях, задаваемых настройками, применяется режим протоколирования записей.

MySQL. Пример настройки репликации

1. В файлы конфигурации серверов (мастера и ведомых) добавить название БД для репликации и уникальный id сервера (например, 1 для главного сервера и 2 – для ведомого):

   ```
   server-id=1
   replicate-do-db=database_name
   ```

2. На главном сервере создать нового пользователя, через которого будет осуществляться репликация, и дать ему соответствующие права.
3. Выполнить настройку репликации на ведомом сервере. Для этого на сервере выполнить команду.

   ```
   CHANGE MASTER TO MASTER_HOST = "localhost", MASTER_PORT=3307, MASTER_USER = "replication", MASTER_PASSWORD = "password";
   ```

4. Выполнить перенос БД главного сервера на ведомый сервер. Для этого с помощью утилиты `mysqldump` создать резервную копию БД, включив в нее информацию о бинарном лог-файле (параметр `--source-data`). Восстановить резервную копию на ведомом сервере.
5. Перезапустить ведомый сервер в режиме «только чтение». Для этого в файл конфигурации ведомого сервера добавить строку «`super_read_only=1`» для того, чтобы осуществлять запись в БД мог только главный сервер репликации.

Механизмы репликации в PostgreSQL

В модели репликации PostgreSQL выделяют два ключевых физических уровневых компонента:

- ведущий сервер (source / master) – сервер, предоставляющий данные;
- подчиненный сервер (replica / slave) – сервер, копирующий себе данные, предоставляемые ведущим сервером.

В PostgreSQL реализовано два типа репликации:

- физическая (потоковая) репликация – трансляция журнала упреждающей записи (Write-Ahead Log Shipping);
- логическая репликация.

Физическая репликация в PostgreSQL

Физическая репликация основана на передаче на реплику изменений в виде записей журнала упреждающей записи. Это очень эффективный механизм, но он требует, чтобы между серверами была **двоичная совместимость** (основная версия сервера, операционная система, аппаратная платформа). Физическая репликация всегда однонаправленна: в ней может существовать только один мастер и произвольное число реплик.

Механизм репликации очень похож на оперативное резервирование данных, с той разницей, что резервная копия, восстановленная на реплике, работает в режиме постоянного восстановления (standby_mode = on) и непрерывно читает и применяет новые сегменты WAL, поступающие с мастера. Реплика постоянно поддерживается в почти актуальном состоянии и в случае сбоя есть сервер, готовый подхватить работу.

Если реплика не допускает клиентских подключений, она называется сервером «теплого резерва». Однако можно сделать и сервер «горячего резерва», тогда в процессе восстановления реплика будет допускать подключения, но только на чтение данных.

Способы доставки журналов:

- *Потоковая репликация*. В этом случае реплика подключается к мастеру по протоколу репликации и читает поток записей WAL. За счет этого при потоковой репликации отставание реплики сведено к минимуму, и даже к нулю при синхронном режиме. При использовании потоковой репликации есть опасность, что мастер удалит сегмент WAL, данные из которого еще не переданы на реплику. Чтобы избежать этого, стоит применять потоковую репликацию вместе с архивом WAL.
- *Репликация с архивом WAL*. При использовании архива специальный процесс на мастере записывает заполненные сегменты журнала в архив. Если реплика не может получить очередную журнальную запись по протоколу репликации, она попробует прочитать ее из архива. В принципе, репликация может работать и с одним только архивом, без потоковой репликации, но в этом случае реплика всегда будет отставать от мастера на время заполнения сегмента.

Физический тип репликации по умолчанию работает в асинхронном режиме, но возможно настроить ведомый сервер на работу в синхронном режиме. Причем в случае нескольких реплик часть могут быть синхронными (резервный мастер), часть – асинхронными (масштабирование чтения).

При асинхронной репликации запрос немедленно выполняется на мастере, а соответствующие данные из WAL передаются к серверам-репликам в фоне. Недостаток асинхронной репликации: при выходе из строя мастера часть данных может быть потеряна, так как они не были переданы на реплики.

При использовании синхронной репликации данные сначала записываются в WAL как минимум одной реплики, после чего транзакция выполняется уже на мастере. Запросы на запись выполняются медленнее в результате возникающих сетевых задержек. Кроме того, чтобы запросы на запись продолжили выполняться в результате выхода из строя одной из реплик, при использовании синхронной репликации рекомендуется использовать по крайней мере две реплики. Но такой подход обеспечивает бо́льшую надежность.

Несколько реплик можно поддерживать без создания дополнительной нагрузки на ведущий сервер используя технологию **каскадной репликации**. Суть этой технологии состоит в том, что одна реплика передает записи WAL другой реплике и так далее. При необходимости иметь копию данных на некоторый момент в прошлом можно сконфигурировать реплику, которая применяет записи WAL через установленный интервал времени.

Логическая репликация в PostgreSQL

Логическая репликация — это метод репликации объектов, данных и изменений в них, использующий «репликационные идентификаторы». В логической репликации используется модель публикаций/подписок с одним или несколькими подписчиками, которые подписываются на одну или несколько публикаций на публикующем узле. Подписчики получают данные из публикаций, на которые они подписаны, и могут затем повторно опубликовать данные для организации каскадной репликации или более сложных конфигураций.

Логическая репликация таблицы обычно начинается с создания снимка данных в публикуемой базе данных и копирования её подписчику, при этом таблицы-приёмники на стороне подписчика должны уже существовать (обычно это делается при первоначальной синхронизации с использованием резервных копий). После этого изменения на стороне публикующего сервера передаются подписчику в реальном времени, когда они происходят. Подписчик применяет изменения в том же порядке, что и узел публикации, так что для публикаций в рамках одной подписки гарантируется транзакционная целостность. Этот метод репликации данных иногда называется *транзакционной репликацией*.

Публикация — это набор изменений, выделяемых в таблице или в группе таблиц. Публикация определяется только в рамках одной базы данных. Публикации могут ограничивать набор публикуемых изменений, выбирая любое сочетание операций из INSERT, UPDATE и DELETE, по умолчанию реплицируются все типы операций модификации данных. Чтобы реплицировать операции UPDATE и DELETE, в публикуемой таблице должен быть настроен «репликационный идентификатор». По умолчанию репликационным идентификатором является первичный ключ таблицы, при необходимости репликационным идентификатором можно назначить другое уникальное поле.

Подписка — это принимающая сторона логической репликации. Узел, на котором определяется подписка, называется подписчиком. В свойствах подписки определяется набор публикаций, данные из которых подписчик будет получать. База данных подписчика может быть публикующей для других баз, если в ней определены собственные публикации. В одной паре публикующий сервер/подписчик могут быть определены несколько подписок, но при этом публикуемые объекты в разных подписках не должны пересекаться. При синхронизации таблицы публикации сопоставляются с таблицами подписчика по полностью заданным именам таблиц. Репликация в таблицы с другими именами на стороне подписчика не поддерживается.

Логическая репликация работает подобно обычным операциям модификации данных в том смысле, что данные на подписчике будут изменены, даже если они изменялись на нём автономно. Если входящие данные нарушают какие-либо ограничения, репликация остановится. Эта ситуация называется конфликтом. При репликации операций UPDATE или DELETE отсутствие данных не вызывает конфликта, так что такие операции просто пропускаются. В случае конфликта выдаётся ошибка и репликация останавливается; разрешить возникшую проблему пользователь должен вручную. Подробности конфликта можно найти в журнале сервера-подписчика. Разрешение конфликта может заключаться либо в изменении данных на стороне подписчика, чтобы они не конфликтовали с приходящим изменением, либо в пропуске транзакции, конфликтующей с существующими данными.

Технически информация об измененных строках извлекается и декодируется из имеющегося журнала WAL на публикующем сервере, а затем пересылается подписчику по протоколу репликации в формате, независимом от платформы и версии сервера. Применение изменений происходит без выполнения команд SQL и связанных с этих накладных расходов на разбор и планирование, что уменьшает нагрузку на подписчика.

Логическая репликация в PostgreSQL

Типовые сценарии использования логической репликации:

- Передача подписчикам инкрементальных изменений в данных, когда они происходят.
- Срабатывание триггеров для отдельных изменений, когда их получает подписчик.
- Объединение нескольких баз данных в одну (например, для целей анализа).
- Репликация между разными основными версиями PostgreSQL.
- Репликация между экземплярами PostgreSQL на разных платформах (например, с Linux на Windows).

PostgreSQL. Пример настройки физической репликации

1. Настроить архивацию WAL (для репликации с архивом WAL). Для этого в конфигурационном файле сервера `postgresql.conf` указать параметры:

   ```
   archive_mode = ON;
   archive_command = 'copy "%p" "PATH\\%f"' # Пример команды архивации для операционной системы Windows
   archive_timeout = 60 # время существования не архивированных данных в секундах
   ```

2. На главном сервере создать нового пользователя, через которого будет осуществляться репликация, и дать ему соответствующие права.
3. Создать копию главного сервера с помощью утилиты `pg_basebackup`.

   ```
   pg_basebackup -p master_port -U replicator -D path_to_slave -Fp -Xs -P -R
   ```

4. В конфигурационном файле главного и ведомого серверов указать команду восстановления из архива WAL.

   ```
   restore_command = 'copy "PATH\\%f" "%p"' # Пример команды для WIndows
   ```

5. Запустить серверы.

Репликация в MongoDB

В MongoDB репликация настраивается путем создания набора реплик.

Набор реплик – это группа серверов с одним первичным узлом, который получает операции записи, и несколькими вторичными узлами, где хранятся копии данных первичного узла.

Если первичный узел дает сбой, вторичные узлы могут выбрать новый первичный узел из их числа. Если используется репликация и сервер выходит из строя, можно получить доступ к данным с других серверов в наборе реплик. Если данные на сервере повреждены или недоступны, можно сделать новую копию данных с одного из других членов набора реплик.

Репликация в MongoDB

Набор реплик основан на двух механизмах: *журнал операций (oplog) и тактовый сигнал (heartbeat).* Журнал операций делает репликацию возможной, а с помощью тактовых сигналов ведется мониторинг состояния и активируется процедура обработки отказа.

Журнал операций представляет собой ограниченную коллекцию в базе *local* на каждом узле, в эту коллекцию записываются все изменения данных, необходимые для воспроизведения операции.

Механизм тактовых сигналов позволяет обеспечить отработку отказа и выбор нового первичного узла. По умолчанию каждый член набора реплик посылает тактовые сигналы всем остальным членам раз в две секунды. Это позволяет в целом отслеживать состояние системы. Если узел перестает отвечать и является первичным, то система автоматически выбирает новый первичный узел из вторичных с самым свежим состоянием.

MongoDB. Пример настройки репликации

1. Инициализация серверов

```
$> mongod --replSet mdbRep --dbpath c:\data\rs1 --port 27017
$> mongod --replSet mdbRep --dbpath c:\data\rs2 --port 27018
$> mongod --replSet mdbRep --dbpath c:\data\rs3 --port 27019
```

2. Настройка набора реплик

```
rsconf = {
    _id: "mdbRep",
    members: [
        { _id: 0, host: "localhost:27017" },
        { _id: 1, host: "localhost:27018" },
        { _id: 2, host: "localhost:27019" }
    ]
}
rs.initiate(rsconf)
```

3. Разрешение чтения из вторичных узлов

```
rs.secondaryOk()
```

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

БЛАГОДАРЮ ЗА ВНИМАНИЕ

Агафонов А.А.
к.т.н., доцент кафедры ГИиИБ