Привет. Меня зовут Алексей Плетнёв и я начальник сектора исследований и разработок компании Базис-Центр. Мы делаем САПР для автоматизации мебельных производств. И, помимо самих инженерных модулей, которые у нас десктопные, мы делаем множество различных веб-сервисов. И всем им нужно где-то хранить свои данные. В нашем случае данные делятся на два типа. Первый - трёхмерные модели и текстуры, которые много весят. Их мы храним на распределённом S3 совместимом хранилище. Второй - различные метаданные, которые весят мало, но их количество достаточно велико. Это данные о пользователях, данные о заказах, о биллинге, статистические данные и много чего ещё. Их, как и многие другие компании, мы храним в базах данных. Но есть одно отличие и заключается оно в режиме работы СУБД. Обычно для таких целей разворачивается один сервер, который принимает всю нагрузку на себя и называется он Мастер. Если его мощностей не хватает, то нагрузку на чтение можно с него снять, добавив один или несколько дополнительных серверов, которые называются Слэйвами. Если требуется обеспечить отказоустойчивость, то можно добавить внешний брокер, который будет следить за состоянием мастера и при выходе оного из строя назначать одного из слэйвов новым мастером, а затем уведомить об этом всех клиентов, если они не поддерживают failover. Такой режим работы называется Master-Slave.

Возникает резонный вопрос – если у меня есть несколько серверов, то почему я не могу работать с любым из них, с которым мне захочется и переключиться на любой другой, когда мне, опять же, этого захочется? А за тем, чтобы это всё корректно работало пускай следит сама СУБД. И такой режим работы есть, называется он Multi-Master. Из названия следует, что любой из серверов может выполнять как читающие, так и пишущие запросы. Наши сервисы используют MySQL, потому далее на его примере я расскажу, как мы настроили и эксплуатируем свой геораспределённый кластер MySQL в режиме Multi-Master.

Для начала схема нашей сети. Наше оборудование размещено в 4 ДЦ на территории России – в Коломне (Московская область), Казани, Самаре и Новосибирске. На слайде среднее время пингов между серверами на этих площадках. Почему именно эти города? Наши пользователи распределены по всей стране. Для комфортной работы с нашими онлайн сервисами трёхмерного моделирования время доступа пользователя до этого сервиса не должно, по нашим оценкам, превышать 50 мс. Соответственно мы хотим, чтобы пользователи из Восточной Сибири ходили на сервера в Новосибирск, а не в Казань. В свою очередь сервисы, работающие в Новосибирске, должны забирать данные с ноды БД, которая тоже крутится в Новосибирске, но при этом данные должны быть синхронизированы с другими нодами на других площадках. Мы развернули наш кластер из 5 нод, 2 из которых находятся в Коломне и по 1 в каждом из оставшихся ДЦ. Расскажу, как мы к этому пришли.

И начну с того, что оба режима – Master-Slave и Master-Master - базируются на процессе репликации. Один из серверов получает данные, формирует набор изменений и рассылает его остальным серверам, которые принимают их и применяют у себя с некоторой задержкой, которая называется лаг репликации. Ключевое отличие в том, что в режиме Master-Slave мастеру неважно, когда именно слэйв применит у себя изменения и применит ли вообще – это проблемы самого слэйва. Мастер сразу после выполнения транзакции отвечает клиенту, что всё ок. Это называется асинхронная репликация. В случае Master-Master пишущая транзакция считается успешной только тогда, когда она применилась на большинстве нод кластера. Это уже синхронная репликация. Если нод 5, то транзакция должна выполниться минимум на 3 и только тогда клиенту вернётся ответ, что всё ок. Данный процесс называется сертификацией транзакции. Это добавит некоторую задержку (тот самый лаг репликации) в выполнение пишущих транзакций, но мы в любой момент времени будем уверены, что не потеряем ни байта наших данных, даже если из ноды, которая приняла транзакцию, внезапно выйдет рабочее тело. Если клиент получил ответ, что транзакция успешно выполнена, значит это действительно так.

В MySQL репликация реализована через плагины. Первый поставляется в стандартной версии MySQL и называется MySQL Group Replication. Второй создан компанией Codership. Называется Galera Cluster и поставляется в составе сборок MariaDB и Percona XTraDB.

Когда мы поняли, что хотим построить multi-master, то начали наши эксперименты со стандартной сборкой MySQL версии 8.0.36 и механизмом Group Replication. Подняли его на трёх серверах в офисе, дали тестовую нагрузку – вроде всё ок. Затем начали разносить ноды в удалённые ДЦ. Снова дали тестовую нагрузку и тут кластер начал постоянно рассыпаться из-за постоянных сработок механизма Flow Control. Расскажу про него чуть подробнее.

Может случиться такая ситуация, при которой поток запросов вырастет на столько, что кластер не будет успевать его обрабатывать. Например, из-за сетевых задержек или разного железа, на котором крутятся ноды. В этом случае включится режим Flow Control. Работает он следующим образом. Нода принявшая запрос, его обрабатывает и отправляет результирующий набор данных для применения на другие ноды. Они попадают в очередь и, соответственно, по очереди применяются. Если какая-то нода не успевает их применять, то очередь на ней растёт. Когда глубина очереди достигает определённого значения, нода сообщает остальным, что она не справляется. Тогда включается тот самый Flow Control, который ограничивает скорость поступления транзакций, т.е. начинает клиентам в ответ на часть запросов возвращать ошибку, что он не успевает их обработать, повторите позже. И в-итоге вообще перестаёт их принимать.

Нас такое поведение, конечно же, не устроило и мы начали искать причины. Основная причина оказалась в том, что механизм MySQL Group Replication не предназначен для георепликации и хорошо работает только в пределах одного ДЦ, а в идеале, в пределах одной серверной стойки. Потому нюансы его настройки я решил опустить в этом докладе, так как от использования такого варианта в проде мы отказались и продолжили эксперименты.

Выбор, что тестировать дальше был между MariaDB и Percona XTraDB с плагином Galera Cluster. MariaDB была больше на слуху, потому начали с неё. Отдельно замечу, что Galera Cluster разрабатывался как альтернатива механизма Group Replication. Его ключевым отличием является поддержка георепликации, которая нам как раз и была нужна.

Если MySQL Group Replication мы настраивали руками, правя конфигурационные файлы, то позже мы нашли очень удобный инструмент от компании SeveralNines под названием Cluster Control. Он позволяет легко и быстро разворачивать, и управлять кластерами и не только MySQL, но и другими. В Community Edition поддерживает до 10 нод в кластере, чего нам вполне достаточно.

Для следующего развёртывания мы просто подготовили 3 пустых машины, закинули на них заранее сгенерированные ssh ключи и указали Cluster Control где разворачивать наш кластер. Примерно 30 минут происходила установка/настройка всего необходимого, после чего мы получили работающий, готовый к экспериментам кластер и начали его нагружать. Сначала всё было вполне себе хорошо, но через несколько дней тестов кластер снова начал рассыпаться. Причём писал при этом в логи самые странные и непонятные ошибки. Зачастую процесс MariaDB просто зависал или падал без объяснения причин. Мы снова начали ковыряться в попытках найти истину.

А истина оказалась проста. MariaDB – форк MySQL, который появился после того, как Oracle стал обладателем кодовой базы MySQL. Опасаясь, что оригинальный MySQL перестанет быть Open Source, сделали MariaDB. И тут их пути разошлись. Во-первых, чем дальше, тем больше начало копиться отличий между форком и оригиналом. Во-вторых, многие в интернете жаловались и жалуются на нестабильность работы MariaDB, её плохое тестирование. Мы в этом убедились сами. Искать каждый день почему вдруг упала очередная нода нам очень не хотелось. А если мы не уверены в стабильности, но мы не будем пускать такое решение в работу.

Осталось протестировать ещё один форк – Percona XTraDB. В отличии от MariaDB, мэйнтэйнеры Percona делают всё возможное, чтобы она была полностью совместима с оригинальным MySQL и втягивают в проект все изменения из оригинала. Помимо этого, Percona отличается более высокой производительностью и масштабируемостью.

Итак, мы удалили очередной кластер и тем же путём через Cluster Control развернули Galera Cluster for Percona XTraDB. Начали тестировать. Пара недель тестирования каких-то явных проблем не выявила и потому мы развернули боевой кластер и начали понемногу переводить приложения на него, параллельно наблюдая за стабильностью и производительностью.

На данный момент все наши приложения используют этот кластер, который находится в работе уже больше года и ни одного сбоя, который бы привёл к остановке кластера или потере данных не произошло. Потому дальше я расскажу про то, как он устроен более глубоко и что нам пришлось доработать, чтобы было удобно им пользоваться.

После того, как мы развернули кластер в режиме Multi-Master, для приложения не имеет значения на какую ноду БД ему идти, чтобы выполнить транзакцию. Все ноды могут выполнить как читающую, так и пишущую транзакцию. Вполне может произойти такая ситуация, когда две транзакции, изменяющие одни и те же данные прийдут на разные ноды кластера одновременно. В этом случае появляется риск получения неконсистентного состояния нашей базы. Чтобы такого не произошло, при развёртывании кластера автоматически включается режим GTID – все транзакции на всех нодах выстаиваются в единую глобальную очередь и на всех нодах выполняются только в этой последовательности.

Как я говорил ранее, все пишущие транзакции проходят через процесс сертификации. Транзакция применяется только тогда, когда большинство серверов кластера подтверждают её исполнение. Как определяется это большинство и как оно может изменяться? Чтобы это понять, давайте рассмотрим процесс первого запуска кластера и его дальнейшей работы.

Для начала давайте рассмотрим возможные состояния любой ноды кластера.

* OPEN – нода только запустилась и пытается установить соединение с кластером
* PRIMARY – нода получила представление о том, как выглядит кластер, кто в него входит и в каком состоянии находится другие ноды
* JOINER – нода говорит кластеру, что она готова к нему присоединиться, но ей нужны актуальные данные
* JOINED – нода получила все актуальные данные, можно впускать её в кластер
* SYNCED – нода стала частью кластера
* DONOR – нода является частью кластера и в настоящий момент отдаёт данные другой ноде, которая в статусе JOINER

Процесс запуска называется Bootstrapping. Мы должны выбрать ноду, которая стартует первой. Она сразу перейдёт в статус SYNCED. Всё, у нас уже есть работоспособный кластер, который может выполнять транзакции. Правда кластером его пока назвать нельзя, так как состоит он всего из одной ноды.

Далее стартует следующая нода. Переходит в статус OPEN и устанавливает соединение с кластером. В режиме multi-master каждая нода устанавливает соединение с каждой другой. После установки соединения нода получает от кластера текущее его состояние и переходит в статус PRIMARY. Теперь ей надо получить все данные, которых у неё не хвататет, т.е. синхронизироваться с кластером. Она переходит в статус JOINER и выбирает произвольную ноду кластера, которая становится донором и качает данные к себе.

Процесс репликации в своей работе использует бинарные логи. Нода, которая исполняет транзакцию, записывает её так же в бинарный лог. Эти изменения передаются другим нодам, которые так же их применяют у себя в той же последовательности. Т.е. бинарный лог – это полный лог изменений, которые произошли с нашими данными. Выполнив его с самого начала, можно получить полную консистентную копию данных. Казалось бы, новая нода должна просто забрать бинарный лог у одной из существующих и проиграть его у себя. Однако не всё так просто. Во-первых, повторение всех транзакций, начиная с самой первой это очень небыстрый процесс. Во-вторых, размеры самого бинлога ограничены и старые данные из него попросту удаляются. Для решения этих двух проблем в Galera Cluster реализован механизм SST, который делает на доноре слепок текущего состояния данных и, быстро перегоняет его но новую ноду. Затем с помощью бинлога доливаются данные, изменившиеся пока шёл трансфер и нода переходит в статус SYNCED, т.е. становится полноценным членом кластера, и сама теперь может стать донором для новых нод.

Процесс сертификации транзакций подразумевает, что более половины от всех узлов кластера должны подтвердить, что они смогли успешно применить её. Каждый раз, когда новая нода присоединяется к кластеру, общее количество узлов меняется и меняется, соответственно и majority группы, т.е. число узлов, которые должны подтвердить транзакцию.

Мы рассмотрели процесс инициализации кластера и добавления в него новых узлов и теперь перейдём к самому интересному. Чтобы всегда оставаться уверенными, что мы не потеряем ни данные, ни их доступность, нам надо понять, к каким рискам мы должны быть готовы.

Первый риск – потеря одной или нескольких нод. Нода может выйти из кластера как штатно, например, при нормальном выключении хоста, на котором она работает, так и нештатно, например, при ненормальном выключении хоста, потери связи и прочее. Что в этом случае произойдёт с кластером? На момент, когда всё функционирует нормально, каждая нода знает о кластере всё, включая общее количество нод, которое в него входит. Если нода выходит штатно, оно сообщает об этом остальным и они перестраивают своё видение кластера, изменяя число нод в нём. Если она отключается или просто теряет связность с кластером, начинает работать другой механизм. Каждая из нод считает, сколько узлов кластера, включая её саму, осталось доступно и сравнивает это число с текущим majority. Если оно оказывается меньше majority, то переходит в статус CLOSED и прекращает принимать запросы, продолжая при этом попытки установить соединения со всеми узлами кластера. Если >= majority, то те ноды, с которыми потерялась связность, исключаются из кластера, а все ноды что остались, пересчитывают majority и начинают использовать новое его значение для последующих расчётов. Что из этого следует? Во-первых, можно без проблем перезагружать один узел за другим, например, для установки обновления ОС. Во-вторых, можно последовательно отключать ноды одну за одной пока не останется всего одна и кластер при этом продолжит функционировать, просто полностью потеряет отказоустойчивость.

Второй риск вытекает из первого. Возможна ситуация так называемого split brain – ситуации, при которой часть нод теряет связность с другой частью. Например, если в кластере 4 ноды, может произойти ситуация, когда между собой будет связность у нод А и B, а также D и E, но не будет связности между этими группами. Что случится? В обоих группах число нод будет меньше majority и все ноды уйдут в статус CLOSED, т.е. весь кластер остановится. Именно поэтому рекомендуется использовать нечётное количество узлов в кластере – так снизится риск его полной остановки. У нас 4 площадки, но на одной из них, где нагрузка выше всего, расположено сразу 2 ноды. Пока ситуаций полной остановки кластера не случалось.

Ну и третий риск – как раз полная остановка кластера, когда все ноды отключились от него и перешли в статус CLOSED. Запустить такой кластер обратно можно только через процедуру bootstrap. В случае multi-master это может быть непросто, потому что не на всех нодах могли быть применены все транзакции. Нам следует найти одну из нод, которая входила в последний majority и где была выполнена транзакция с самым большим порядковым номером и забутстраппить её. Затем стартовать другие ноды. Вообще, инструмент Cluster Control умеет делать автоматическое восстановление кластера, но уже не в Community Edition и не всегда это может сработать корректно, потому лучше такой старт производить вручную.

И в конце расскажу немного про пару хитростей, которые мы применили в архитектуре нашей сети для того, чтобы не нужно было вообще модифицировать как-то наши сервисы для работы с кластером. Итак, мы имеем 4 локации и кластер из 5 нод. Как сделать так, чтобы сервис ходил всегда в ближайшую ноду кластера в своём же ДЦ, а в случае её отказа уже в другой ДЦ? Обычно для этого используют различные прокси типа ProxySQL, HAProxy и т.п. Нам всего лишь нужно проксировать TCP соединение до ближайшей ноды или до любой живой, если ближайшая недоступна. Используемые нами пограничные маршрутизаторы, как и многие другие, имеют встроенные механизмы балансировки нагрузки, включай failover и weighted round robin. Балансировка работает как раз как TCP прокси. Потому мы решили не городить дополнительные точки отказа в виде промежуточных программных проксей, а просто каждому балансировщику указали все ноды, а также их веса. На внешнем интерфейсе каждого маршрутизатора мы повесили один и тот же ip – 192.168.254.254. Теперь мы можем упаковать наш сервис в контейнер, указав в качестве IP СУБД 192.168.254.254 и свободно запускать этот контейнер в любом из наших ДЦ – он всегда прийдёт к ближайшему маршрутизатору, который установит соединение с ближайшей нодой. Всё просто и красиво, но есть одна проблема. Программные прокси умеют определять состояние ноды и устанавливать подключение только в случае, если она находится в статусе SYNCED или DONOR. В остальных случаях нода считается неживой и клиенты на неё не отправляются. Обычный же TCP балансировщик просто отправляет ноде пакет TCP SYN и считает её полностью работоспособной, если получит в ответ TCP ACK, игнорируя при этом факт того, является ли нода уже полноценным членом кластера, или нет. Чтобы это обойти, я написал небольшой скрипт, который по расписанию запускается на каждой ноде кластера каждый 30 секунд и проверяет её состояние. В случае, если нода находится в нужном состоянии, в iptables добавляется правило перенаправления трафика с порта 3307 на стандартный для MySQL 3306. В балансировщиках указан порт 3307. Таким образом, если с нодой всё в порядке, балансировщик может подключиться к этому узлу на порт 3307 и начать устанавливать с ней соединения. Если что-то происходит, правило удаляется и балансировщик перенаправляет клиентов к другим нодам.

Проект опубликован на гитхабе, пользуйтесь, вот ссылка.

И напоследок несколько ограничений применительно к MySQL в режиме multi-master:

1. Как и в случае с другими кластерами, желательно, чтобы железо каждого узла было одинаковым, так как производительность кластера будет равна производительности самого медленного узла.
2. Поддерживается только движок InnoDB (и его улучшенная версия XtraDB), так как только он является транзакционным.
3. У каждой таблицы обязательно должен быть первичный ключ.
4. Автоинкрементные поля увеличиваются с шагом, равным количеству нод в кластере и значение счётчика нельзя трогать руками.

Как итог, хочу сказать, что эксплуатация MySQL кластера в multi-master режиме не так страшна, как может показаться на первый взгляд.