Асинхронная репликация MySQL без цензуры

Олег Царёв

highload++ 2014

Конференция разработчиков высоконагруженных систем

Вы любите гномиков?

Гномики опасны

Знакомимся

- Писал OLAP, OLTP и NoSQL базы данных (QD, MySQL, SciDB)
- Помогаю с эксплуатацией MySQL в Mail.Ru Target
- Занимаюсь вопросами производительности софта
- Подслушиваю в курилке, что говорят умные люди
- Борюсь с гномиками в базах данных
- Если коллеги видят гномика — я его убиваю и провожу вскрытие
- Писал репликацию, настраивал репликацию, объяснял репликацию

Повестка дня

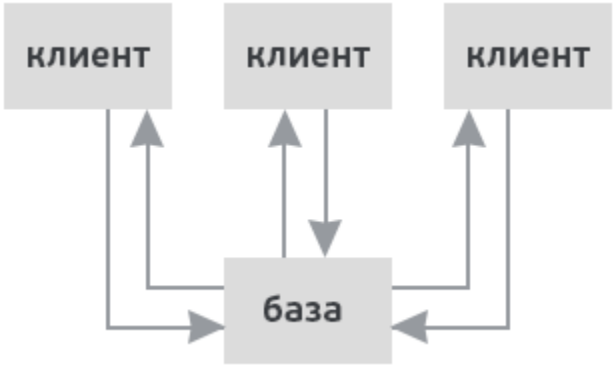
- Что такое репликация
- Как она используется
- Журнал СУБД и связь с репликацией
- Типы репликаций (физическая, логическая, query-based)
- Плюсы и минусы разных подходов
- Архитектуры репликаций в PostgreSQL и MySQL
- Архитектурные проблемы репликаций в MySQL. Можно ли было их избежать?
- Что нового в MySQL 5.6 / 5.7

Disclaimer

...одно из положений теории систем гласит: систему невозможно оптимизировать по двум независимым параметрам одновременно; в частности, добиваясь целостности рисуемой картины, неизбежно приходится жертвовать ее детальностью, или наоборот.

Кирилл Еськов, «История земли и жизни на ней»

Как видит базу инженер



- Внутри происходит магия

Две базы лучше чем одна

Репликация, это...

- Репликация — распространение изменений между базами
- Асинхронная репликация — отложенное распространение изменений
- В MySQL, строго говоря, существует лишь асинхронная репликация
- Однако есть ещё semisync, galera, NDB Cluster, MMM, Tungsten
- Но широко используется лишь асинхронная
- В PostgreSQL есть асинхронная (из коробки) и Trigger-based (внешняя)

Репликация это не бекапы

- Репликация не является резервной копией
- Репликация **никогда** не является резервной копией
- Даже если сильно хочется — репликация не является резервной копией
- Резервная копия — состояние в прошлом, на которое можно вернуться
- DROP DATABASE very_important_database; <== **и что делать?**
- **Помнить: «репликация не является резервной копией»**

Пусть master упал

У нас остался slave!

Резервные копии

- Репликация — не резервная копия **(вы запомнили?)**
- Как делать резервные копии не мешая приложению?

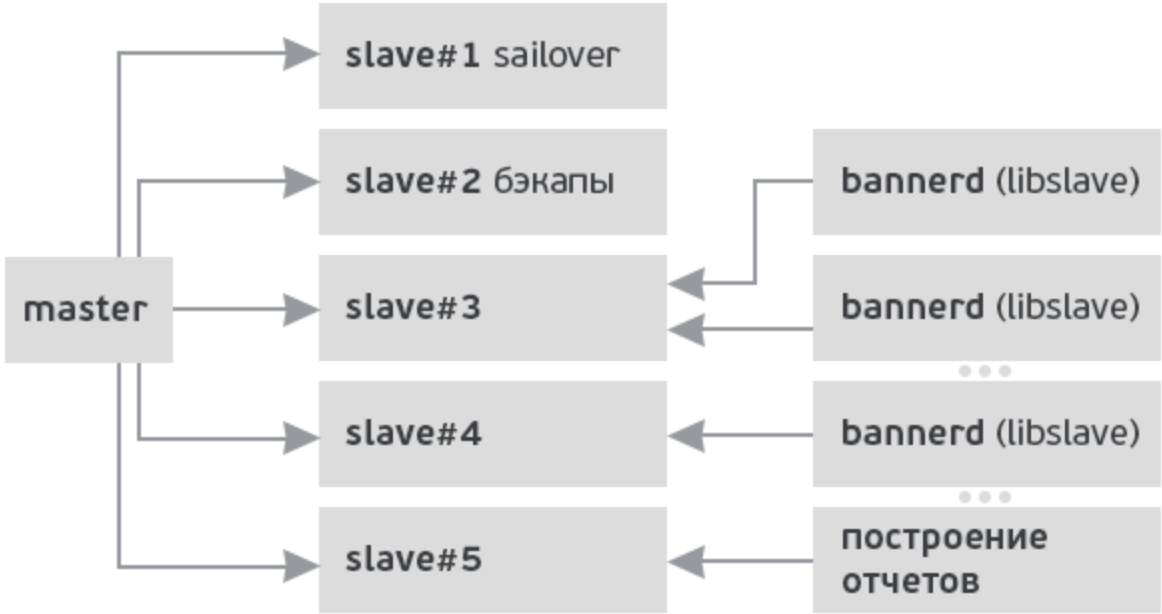
Если мастер задыхается

- Можно делать долгие запросы (отчёты? статистика?) на slave

Демон как MySQL-slave

- Демон хочет получать/обрабатывать изменения базы
- http://habrahabr.ru/company/mailru/blog/219015/
- У нас таких серверов десятки

Собираем всё вместе



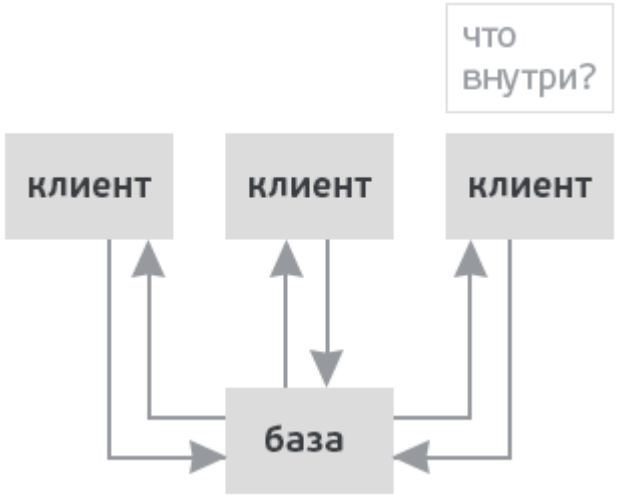
*Структура проекта Mail.Ru Target

Mail.Ru Target

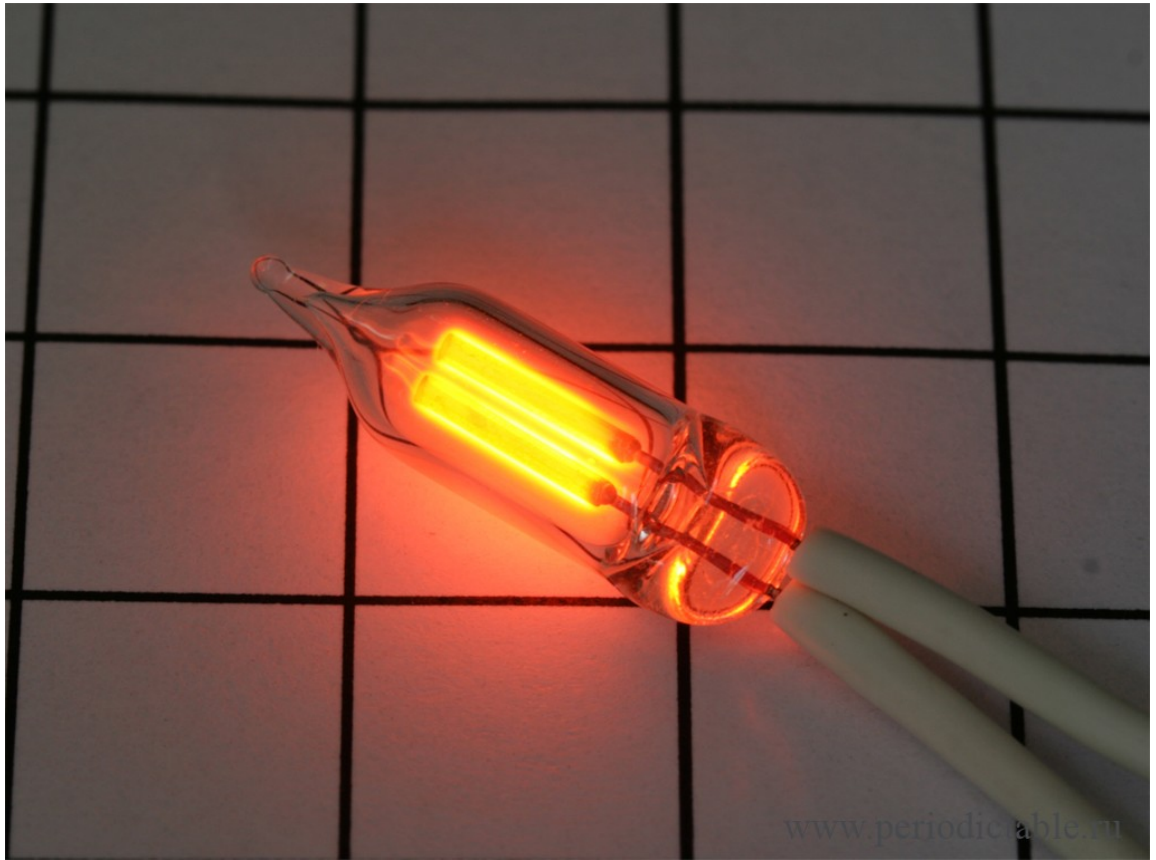
- Размер базы около **1.5 терабайт**
- ~6 000 запросов в секунду на master
- ~15 000 запросов в секунду на репликах
- Более ста различных серверов приложений работающих с базой
- За 3+ года года работы не лежали ни разу
- Авторы libslave, Федор Сигаев, Костя Осипов по соседству
- Ещё у нас есть печеньки и спортзал
- В общем, без репликации нам никак

Залезем в шкуру автора

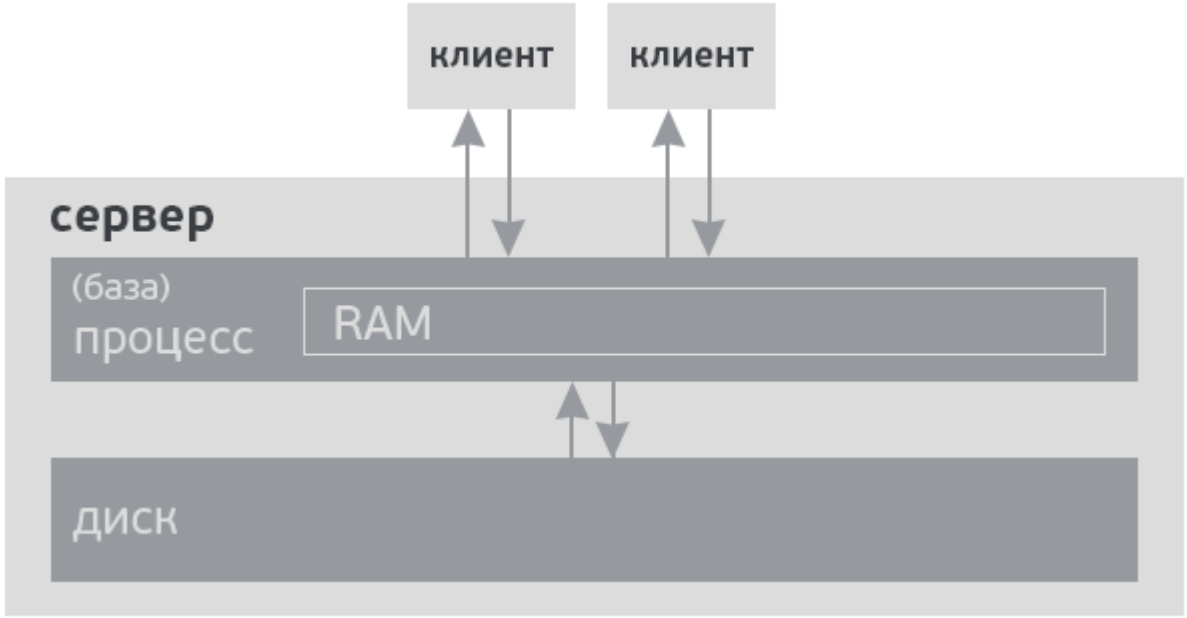
- Как именно база гарантирует принцип «всё или ничего»?
- Как избежать коллизий и гонок? <= за пределами доклада
- Как пережить падение питания?

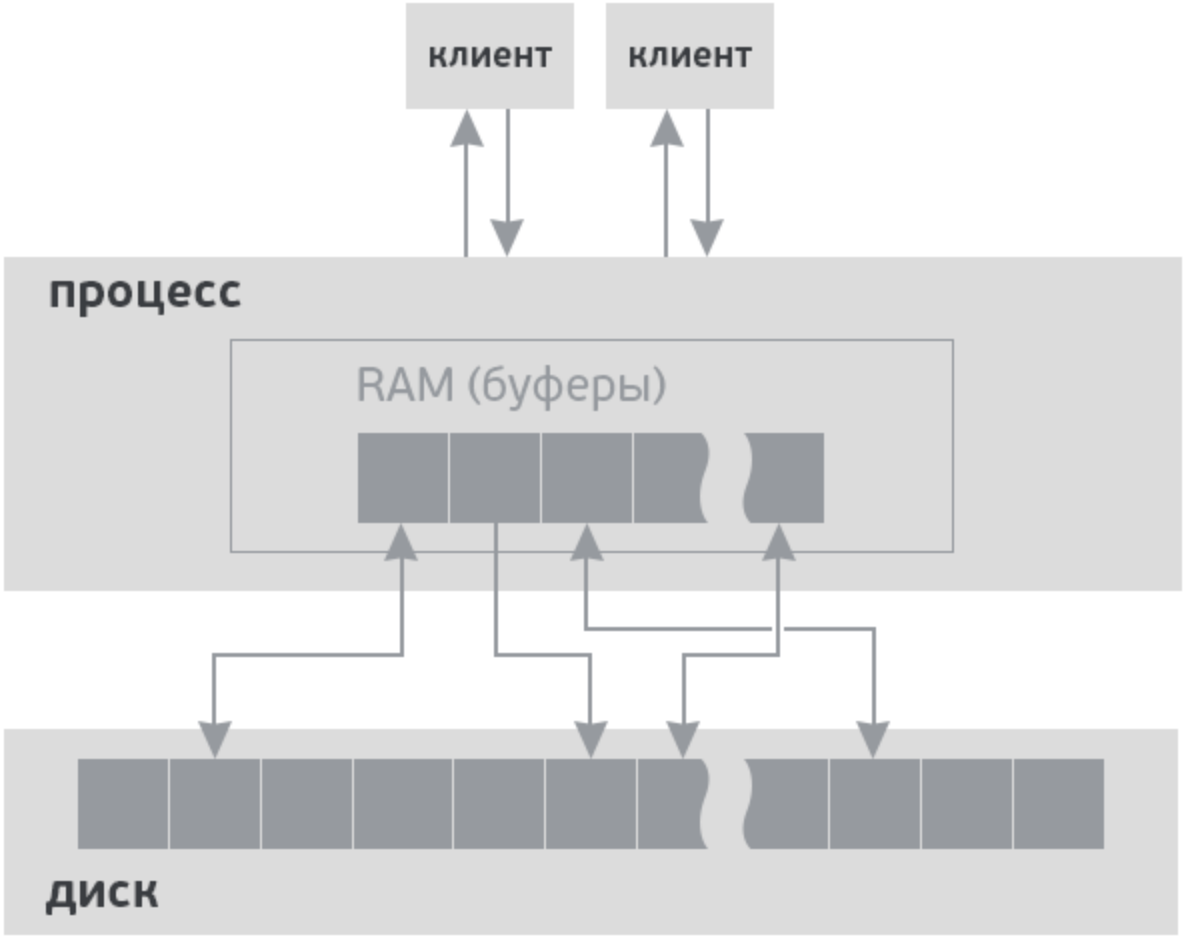


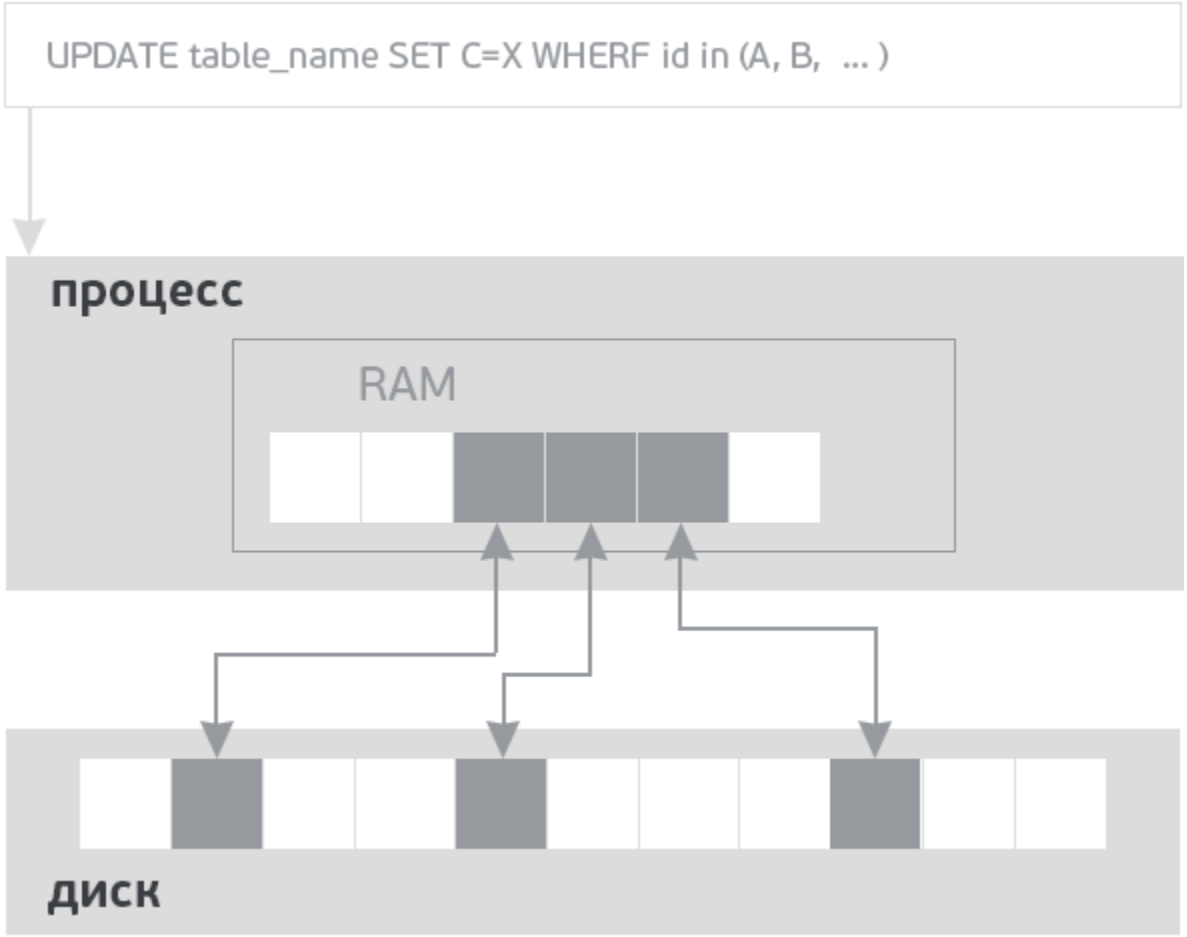
А что у ней внутри?



Внутре у ней неонка

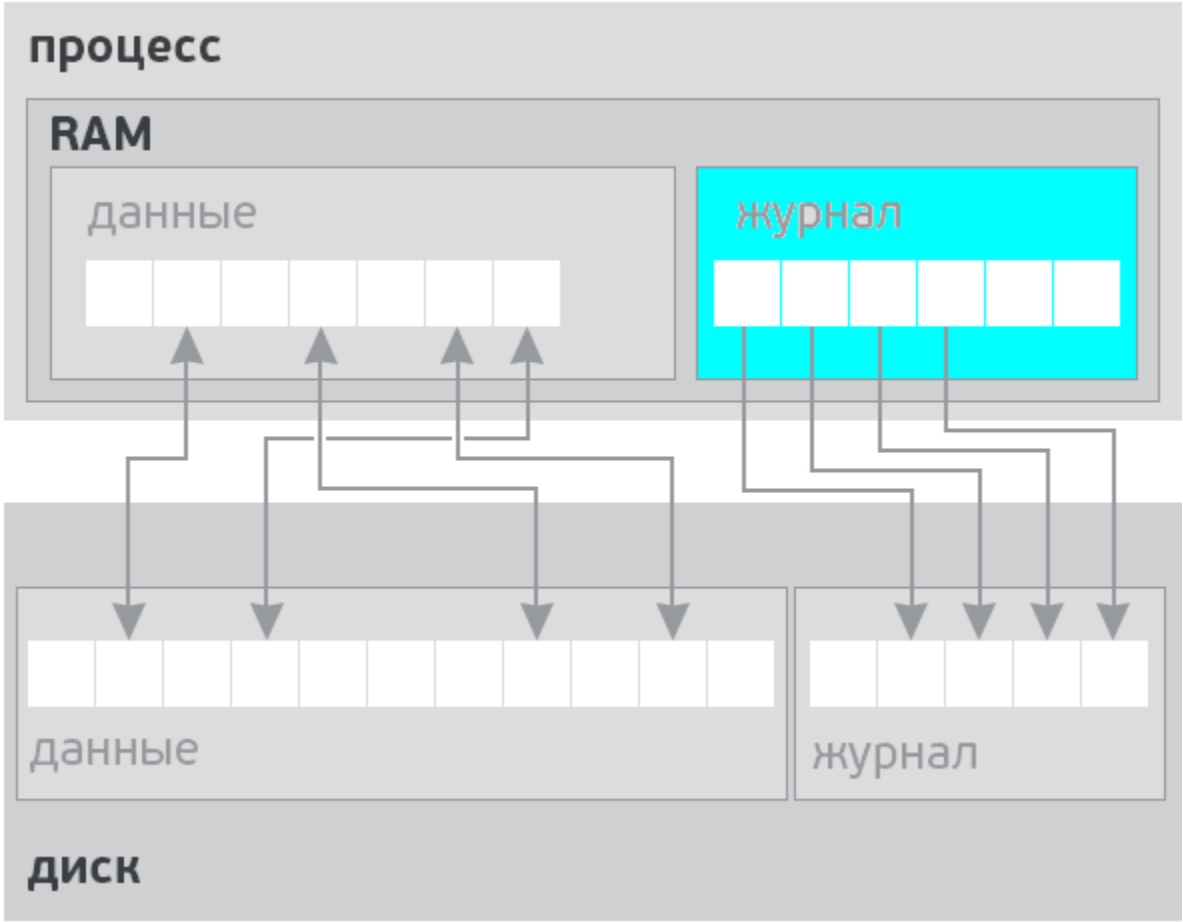


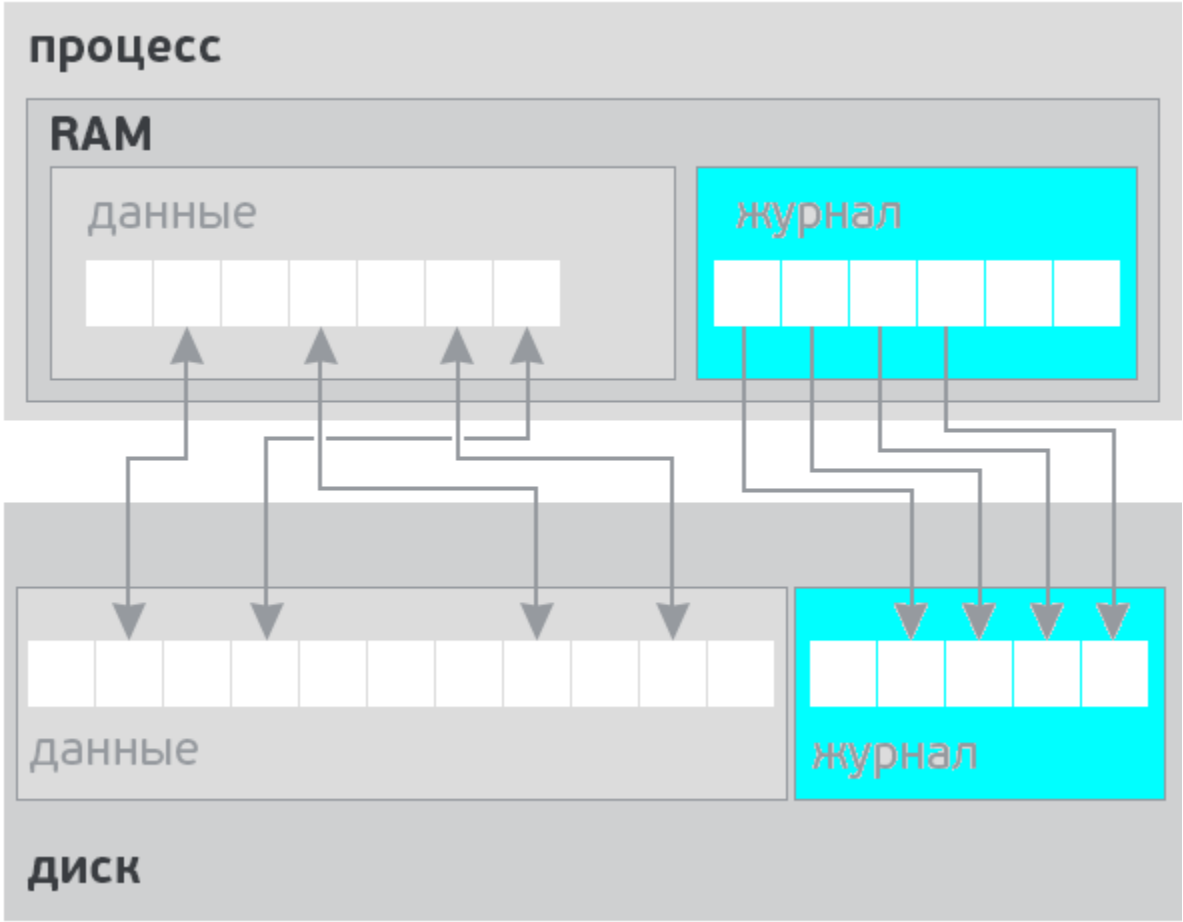


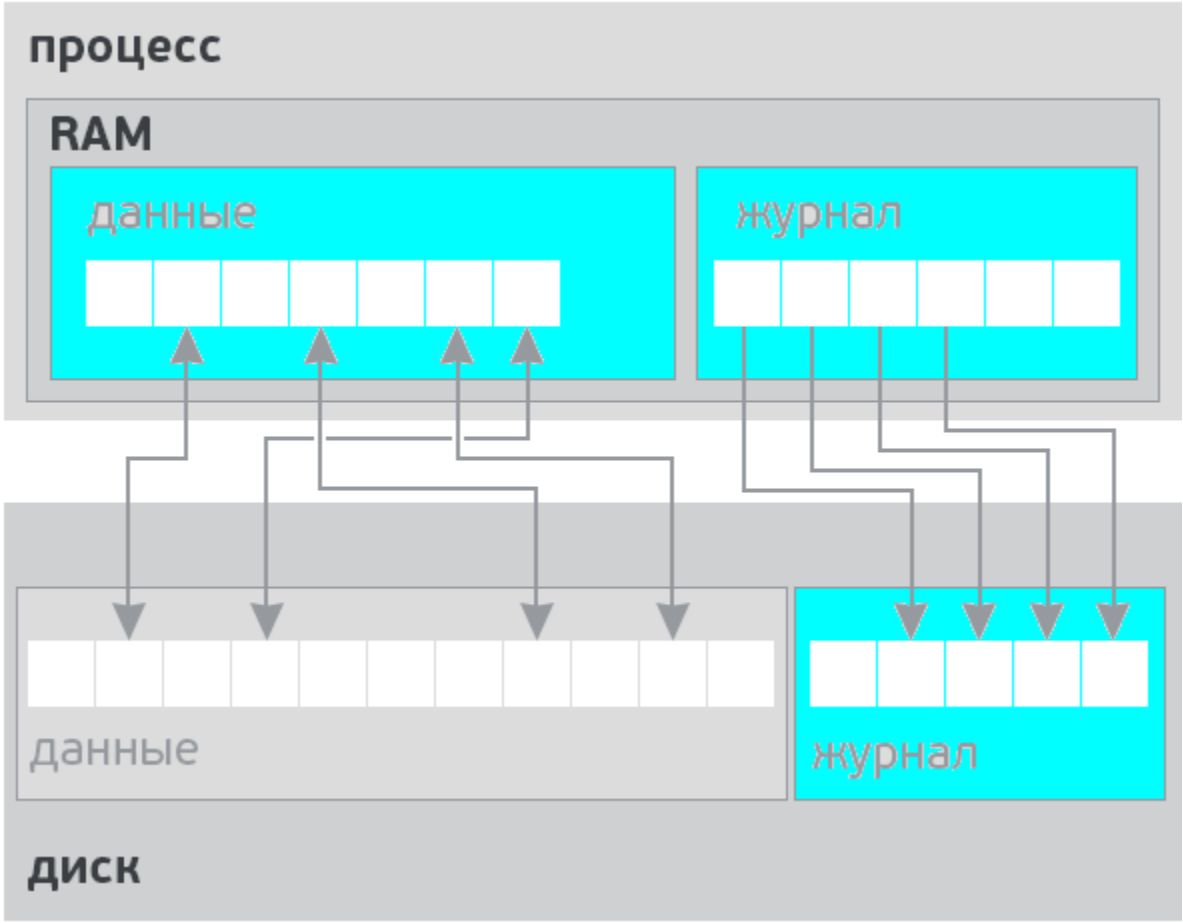


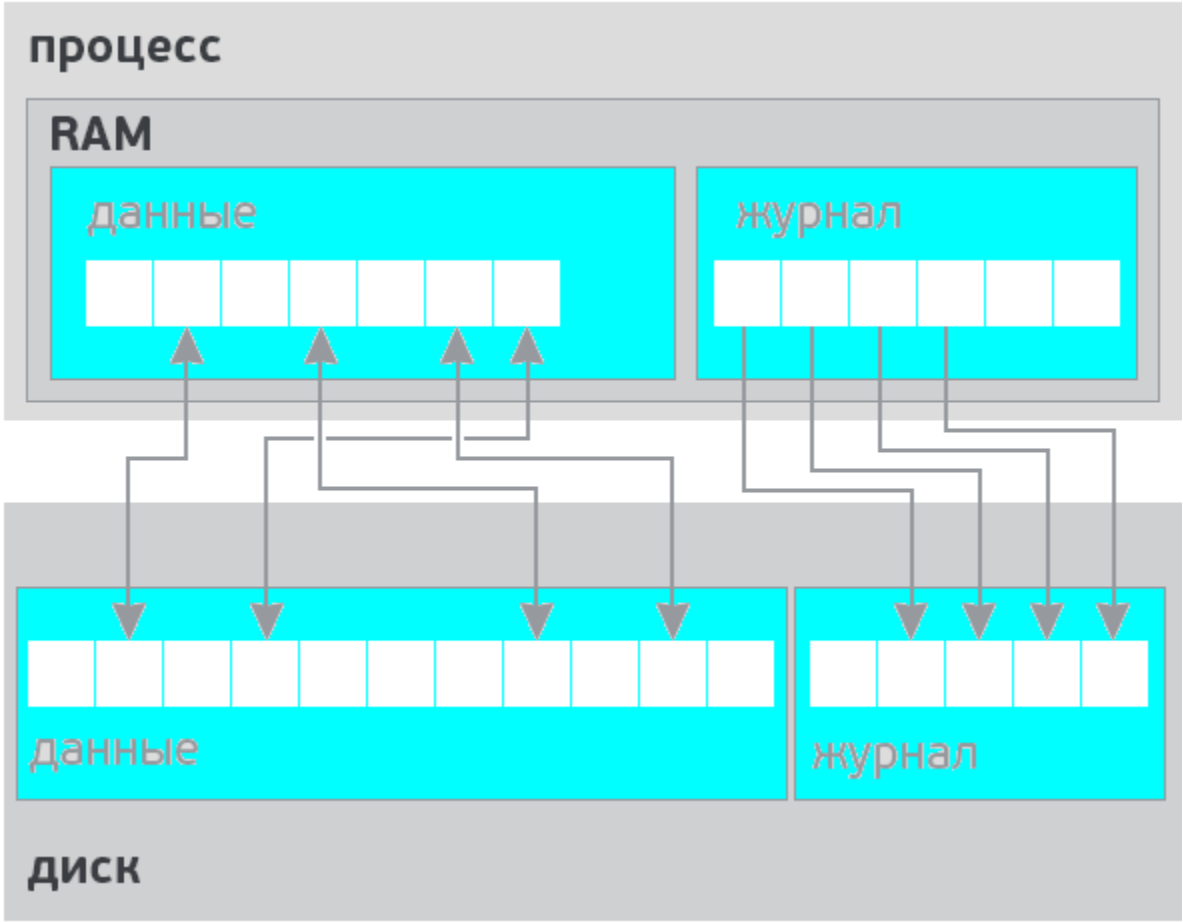
Проблемы

- Принцип «всё или ничего» не выполняется при записи диска
- Даже с оперативной памятью научились буквально только что
- Питание у сервера может упасть в **любой** момент
- Сеть тоже — против бульдозера оптике нечем возразить
- Нужно найти способ надёжно писать данные в ненадёжной среде
- Как ни странно, ответ подсказали бухгалтеры









Учтите

- Это называется **Point In Time Recovery** или **PITR**
- Научная работа на эту тему зовётся **ARIES** http://en.wikipedia.org/wiki/Algorithms_for_Recovery_and_Isolation_Exploiting_Semantics
- Есть умная книжка **Transactional Information Systems**

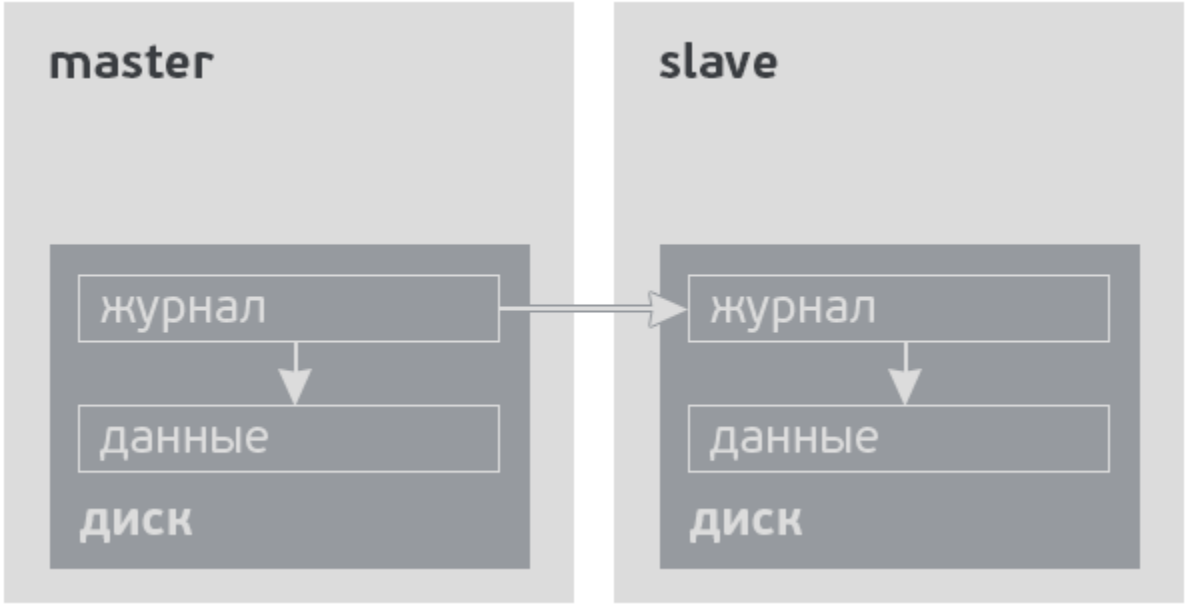
  http://www.amazon.com/Transactional-Information-Systems-Algorithms-Concurrency/dp/1558605088
- Про MySQL читайте: http://www.percona.com/blog/
- Про PostgreSQL можно проштудировать: http://www.pgcon.org/2012/schedule/track/Hacking/408.en.html
- **Всё сложней чем я описал** (упростил для наглядности)

Ключевые вопросы

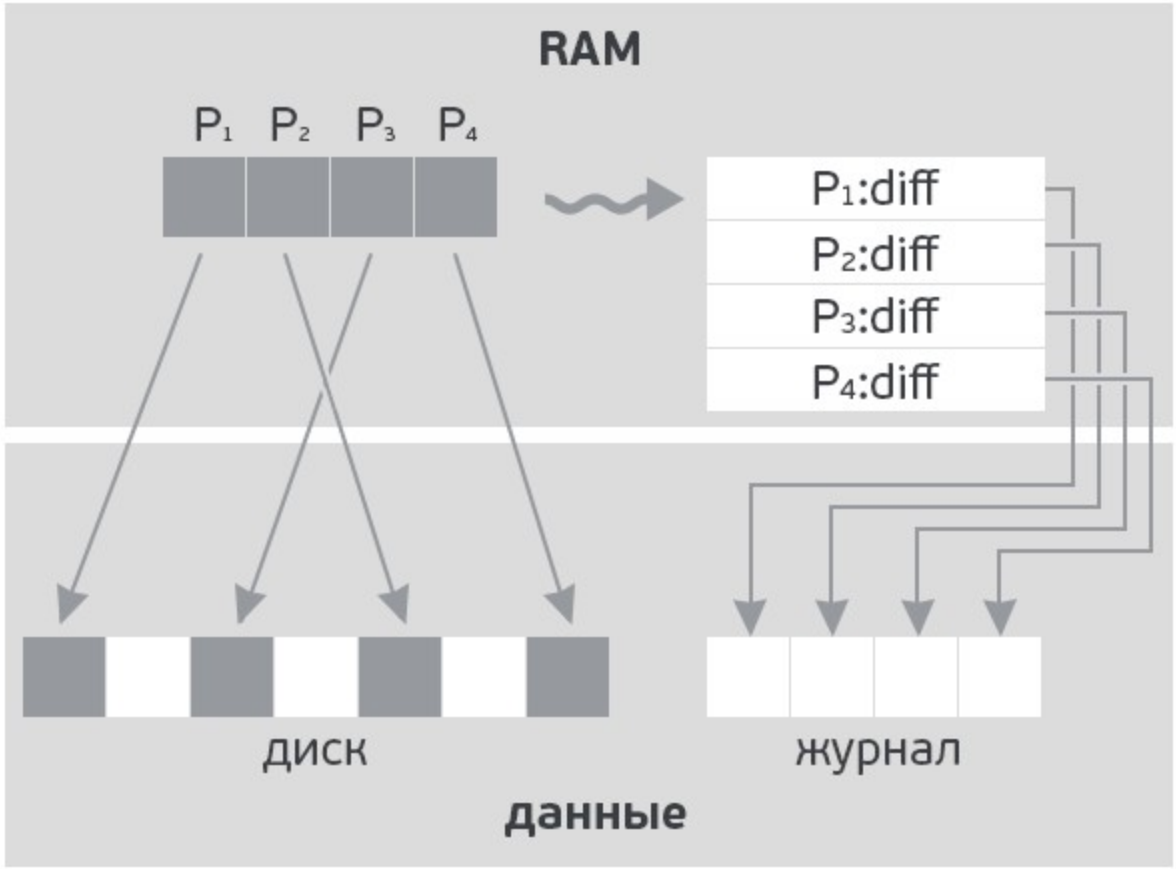
- Как организовать журнал?
- Как уменьшить объём записи на диск?
- Как уменьшить количество fsync?
- Причём тут репликация?

Copyright http://svpsychology.ru/ulibka-do-ushey/

Репликация: прямой путь

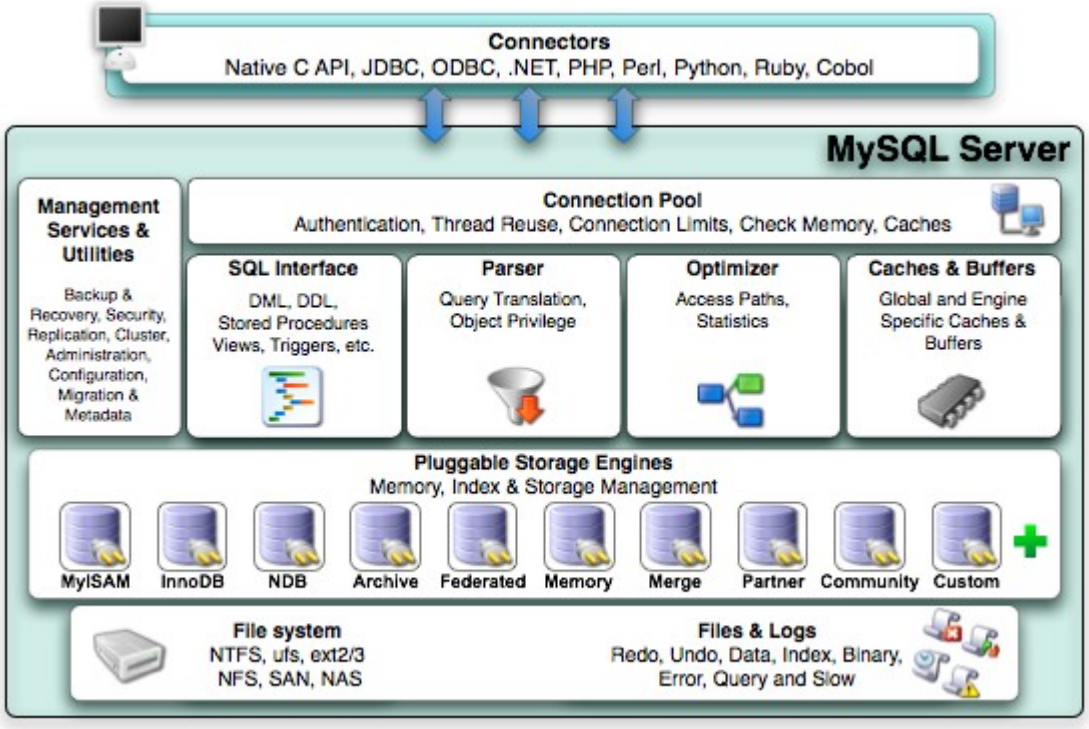


PostgreSQL: WAL



Сколько журналов в MySQL?

Архитектура MySQL



Где журнал?

Изучим историю

- PostgreSQL Hackers 2004 год
- http://www.postgresql.org/message-id/200407260025.19914.peter_e@gmx.net

Изучим историю

In mysql, we can wrote a create table like

CREATE TABLE t (i INT) ENGINE = INNODB||BDB|;

where the storage engine is the innodb one.

This allow to have differents kind of storage format, and allow to easly implements memory table or remote table.

I try to make the same thing for postgresql but i do not understand where the logical storage engine is in the source code.

May i have somme help to find it .

http://www.postgresql.org/message-id/40FBCF57.3050709@limsi.fr

Изучим историю

Well, it certainly could make sense to have different storage engines

for different access patterns. ...

http://www.postgresql.org/message-id/200407260025.19914.peter_e@gmx.net

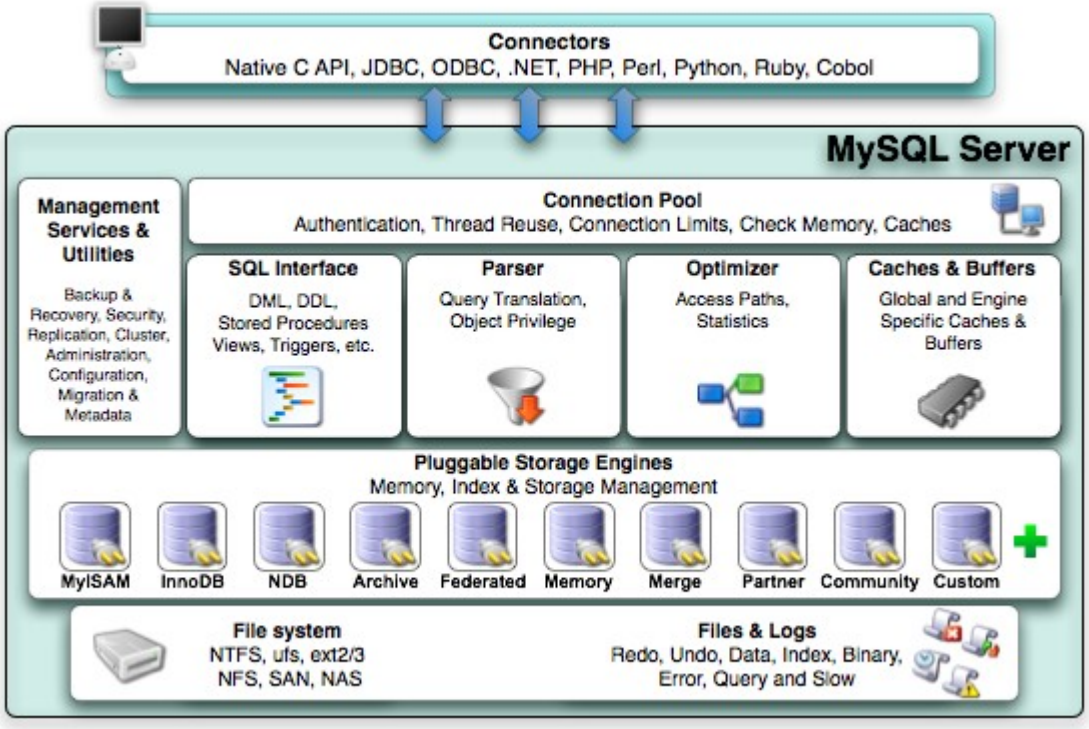
Изучим историю

The problem is that many storage management systems ... often do their own WAL and PITR. Some do their own buffer management, locking and replication/load management too.
So, as you say, its hard say where an interface should be abstracted.

http://www.postgresql.org/message-id/Pine.LNX.4.58.0407261447400.18278@linuxworld.com.au

Gavin Sherry

Архитектура MySQL



История

- MySQL придумали концепт Storage Engine (основной — MyISAM)
- MyISAM не имеет журнала (так тоже **можно**! но **не нужно**)
- MySQL сделали репликацию
- Для репликации сделали имитацию журнала — binary log
- MySQL подарили InnoDB
- У InnoDB свой журнал
- Какой журнал передавать с master на slave?

Архитектурная ошибка

- MySQL реализовал отдельный тип журнала: binary log
- **Binary log не используется для Point In Time Recovery**
- **InnoDB Undo/Redo Log не используется в репликации**

Так что там с журналом?

- InnoDB **Undo/Redo Log** похож на PostgreSQL WAL
- **Binary log** — отдельный журнал с шахматами и феями
- Форматы binary log:
  - **SBR** — Statement-based replication
  - **RBS** — Row-based replication
  - **Mixed** format logging
- Но раз уже его сделали, давайте разбираться

Statement-based

- Журнал — файл с последовательно записанными событиями
- Каждой событие — одна транзакция
- Транзакция — SQL запросы, от *BEGIN* до *COMMIT*
- В начале события идут два поля
  - use dbname; <== *ведь у нас может быть несколько баз*
  - timestamp **времени завершения транзакции на master**

Statement-based

Row-based

- Журнал — последовательно записанные события
- Событие содержит **BEFORE** и **AFTER** образы
- Из каждой изменённой таблицы в образ попадает набор строк
- Как именно:
  - DELETE — строка есть в BEFORE image
  - INSERT — строка есть в AFTER image
  - UPDATE — строка есть в обоих
  - ALTER — как в statement-based
- Если на пальцах: это такие бинарные патчи

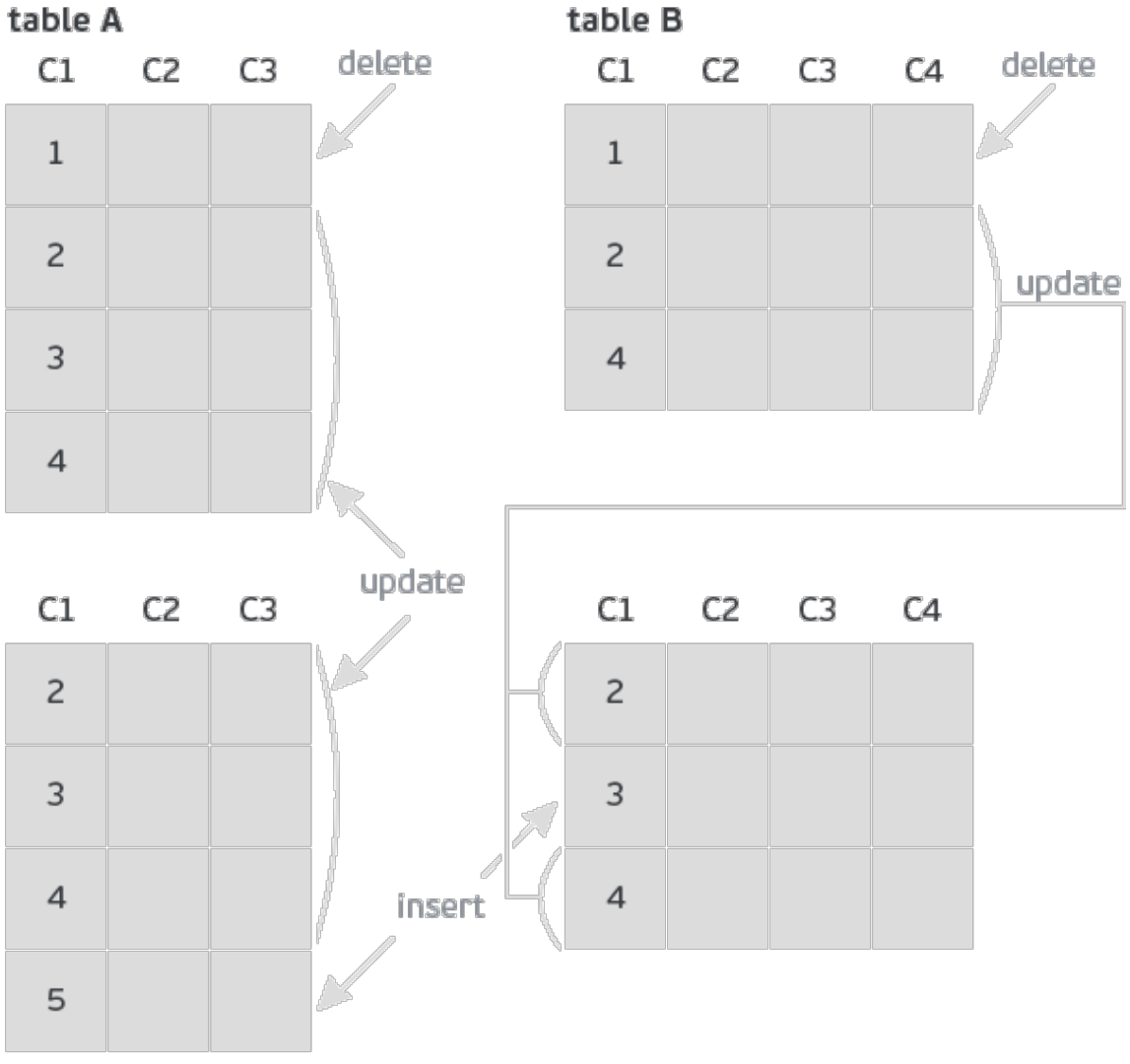


table A
C1
C2
C3
delete
1
2
3
4
table B
C1
C2
C3
C4
delete
1
2
4
update
update
C1
C2
C3
2
3
4
5
insert
C1
C2
C3
C4
2
3
4


table A
C1
C2
C3
delete
1
2
3
4
table B
C1
C2
C3
C4
delete
1
2
4
update
update
C1
C2
C3
2
3
4
5
insert
C1
C2
C3
C4
2
3
4


table A
C1
C2
C3
delete
1
2
3
4
table B
C1
C2
C3
C4
delete
1
2
4
update
update
C1
C2
C3
2
3
4
5
insert
C1
C2
C3
C4
2
3
4

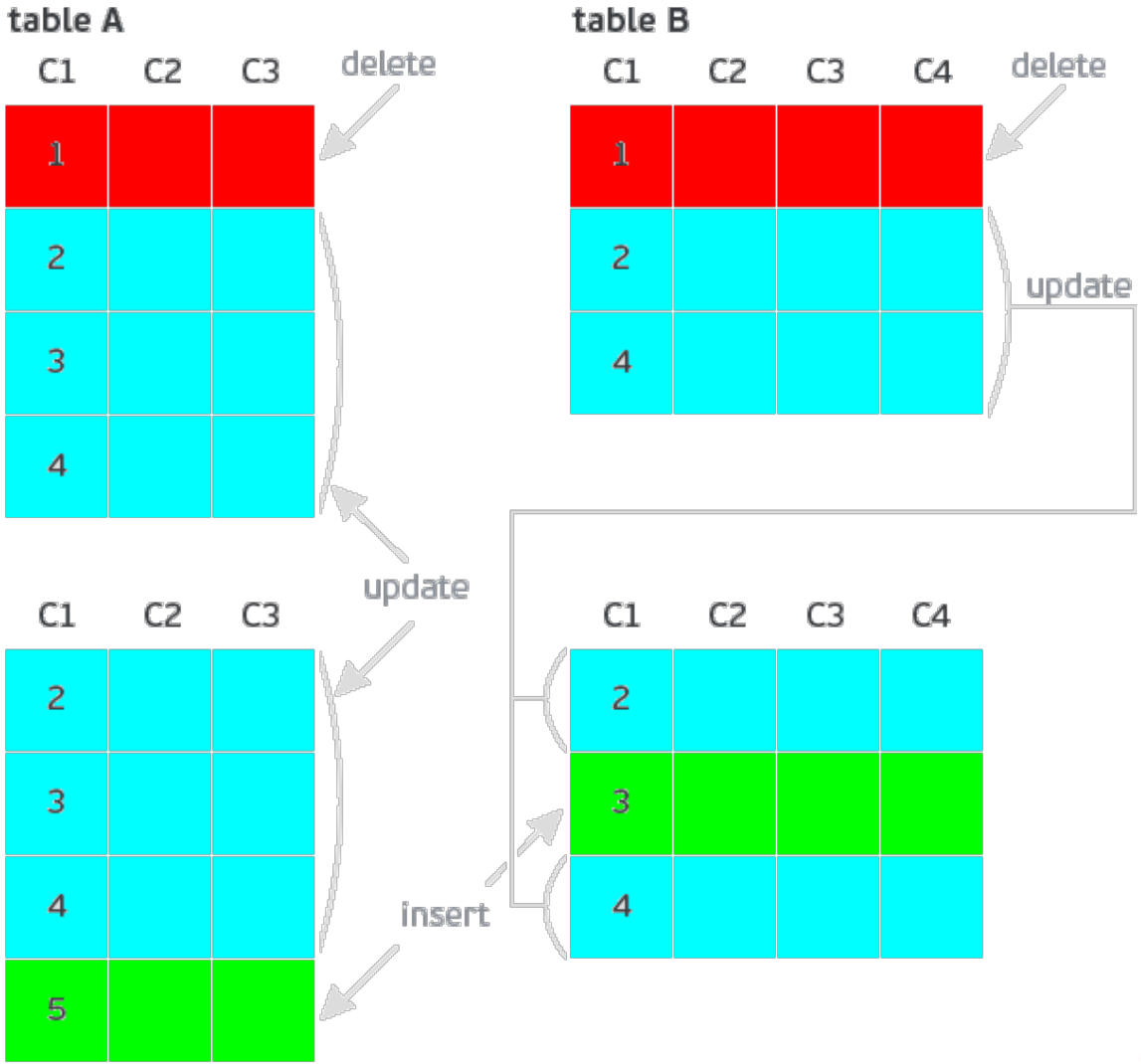


table A
C1
C2
C3
delete
1
2
3
4
table B
C1
C2
C3
C4
delete
1
2
4
update
update
C1
C2
C3
2
3
4
5
insert
C1
C2
C3
C4
2
3
4

Row-based

- Обратите внимание: образы содержат **все** колонки таблицы
- До MySQL 5.5 включительно
- MySQL 5.6: **binlog_row_image**
  - full — все колонки
  - minimal — только затронутые
  - noblob — full кроме blob, если blob не менялись
- Это серьёзное улучшение

Mixed-based

- Это такая statement-based которая иногда row-based
- Таким образом хотели решить отдельные проблемы
- С версии **5.1.12** по **5.1.29** это даже был формат по умолчанию
- Широко не используется

В чём сила?

Реляционные таблицы

- Кто в зале знает определение?

Реляционные таблицы

- Часто думают, что реляционная таблица — это массив

Реляционные таблицы

- Часто думают, что реляционная таблица — это массив
- Некоторые даже думают, что это двумерный массив

Реляционные таблицы

- Часто думают, что реляционная таблица — это массив
- Некоторые даже думают, что это двумерный массив
- Но таблица — **гомогенное мультимножество кортежей**
- **Кортеж** — набор упорядоченных типизированных значений
- **Гомогенное** — они одинакового типа
- **Мульти**множество — могут быть дубликаты
- Мульти**множество** — порядок элементов не задан
- **Строка** — кортеж, **таблица** — их гомогенное мультимножество

Хаос порождает порядок

- Порядок выдачи строчек в запросе не определён
- Окей, можно написать ORDER BY
- Даже с ORDER BY для повторяющихся ключей порядок не задан
- Понимание таблицы как «массива» не помогает, а мешает
- **Таблица - «гомогенное мультимножество»**
- Запомните и никогда не забывайте

Statement based replication

ALTER TABLE **t** DROP COLUMN **old_key**;
ALTER TABLE **t** ADD COLUMN **key** INT PRIMARY KEY AUTO_INCREMENT;

Statement based replication

ALTER TABLE **t** DROP COLUMN **old_key**;
ALTER TABLE **t** ADD COLUMN **key** INT PRIMARY KEY AUTO_INCREMENT;

Хьюстон, у нас проблема!

Проблема в том, что...

- Запросы на master и slave выполняются по-разному
- Точнее, в разном порядке
- Физический порядок строчек на master и slave будет разный
- Значения в колонке **key** на master и slave будут различными
- Мы на это тоже наступили
- DBA! Приглядывайте за админами и особенно разработчиками
- Но у нас на master и slave было разное число строк
- Но это совсем другая история...

Statement-based и ALTER

```
CREATE TABLE t2 LIKE t1;

ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;

INSERT INTO t2

SELECT * FROM t1 ORDER BY col1, col2;
```

http://dev.mysql.com/doc/refman/5.5/en/replication-features-auto-increment.html

Логическая и физическая

- **Физическая** репликация работает со страницами
- **Логическая** репликация работает с кортежами
- PostgreSQL WAL, InnoDB Undo/Redo Log — **физическая**
- Row-based binary log — **логическая**
- Statement-based binary log — **недоразумение**
- Statement-based binary log — это даже не журнал
- Всё так по историческим причинам

Репликация и индексы

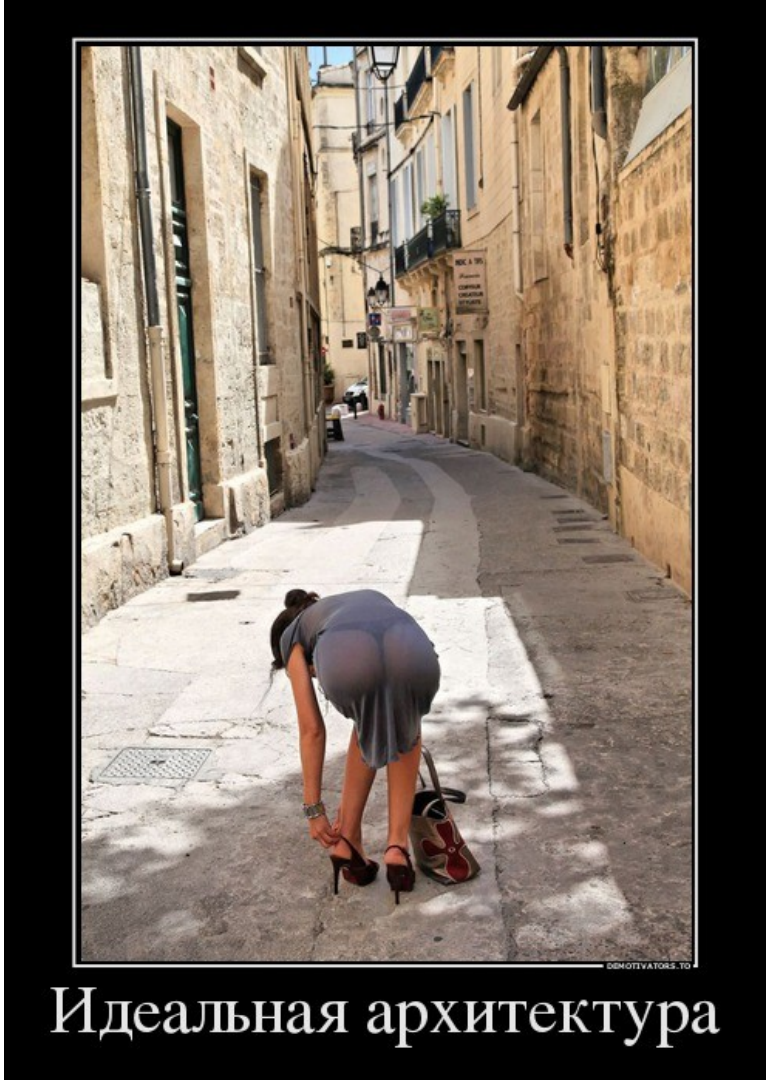
- **Логическая** репликация (**row-based**) «не знает», как хранятся данные на диске
- При применении событий из row-based binlog нужно найти в таблице и индексах строчки, которые мы обновляем
- **Производительность логической репликации зависит от индексов на slave**
- **Физическая** репликация *обычно* **IO-bound**
- **Логическая** репликация *обычно* **CPU-bound**

Репликация и CPU (Slave)

- **Statement-based** репликация так же плоха, как **row-based**
- Точнее она **хуже** (с точки зрения потребления процессора)
- Выполнение запроса на slave также трудоёмко, как выполнение запроса на master

Репликация и IO (master)

- PostgreSQL пишет в **два** места: **хранилище данных** и **журнал**
- MySQL InnoDB master пишет в **три** места:
  - **Хранилище** (tablespace)
  - **Журнал** (undo/redo log)
  - **Binary log**
- **MySQL row-based** репликация **в полтора раза** хуже **PostgreSQL** по объёму записи (master)



Идеальная архитектура

Репликация тормозит...

Репликация тормозит...

- Как найти причину?

Мы делаем так

Диагностика

- В 5.6 / 5.7 есть SLAVE PERFORMANCE SCHEMA
- С 5.5 приходится поднимать git log для puppet
- Иногда это не помогает
- Тогда я беру approvement на пытки у технического директора (спасибо, Дима Кирноценский!)
- Но даже пытки не всегда помогают
- Можно посмотреть slow query log (но это если statement based)
- Часто нету топа медленных запросов — просто их много
- В общем, всё грустно

Но и это ещё не все

Репликация и master

- Сильная сторона асинхронной репликации — отсутствие отрицательной обратной связи
- http://dev.mysql.com/doc/refman/5.5/en/innodb-parameters.html#sysvar_innodb_locks_unsafe_for_binlog
- **sysvar_innodb_locks_unsafe_for_binlog=0**
- Чтобы statement-based корректно работала, нам нужно запретить параллельную вставку новых записей на master
- Абстракции протекли: InnoDB (storage engine layer) вынужден знать про репликацию (MySQL-layer)

Параллелизм

- Master выполняет запросы **параллельно**
- В журнал и/или лог запросы пишутся **последовательно**
- Можно ли выполнять запросы параллельно?

Параллелизм

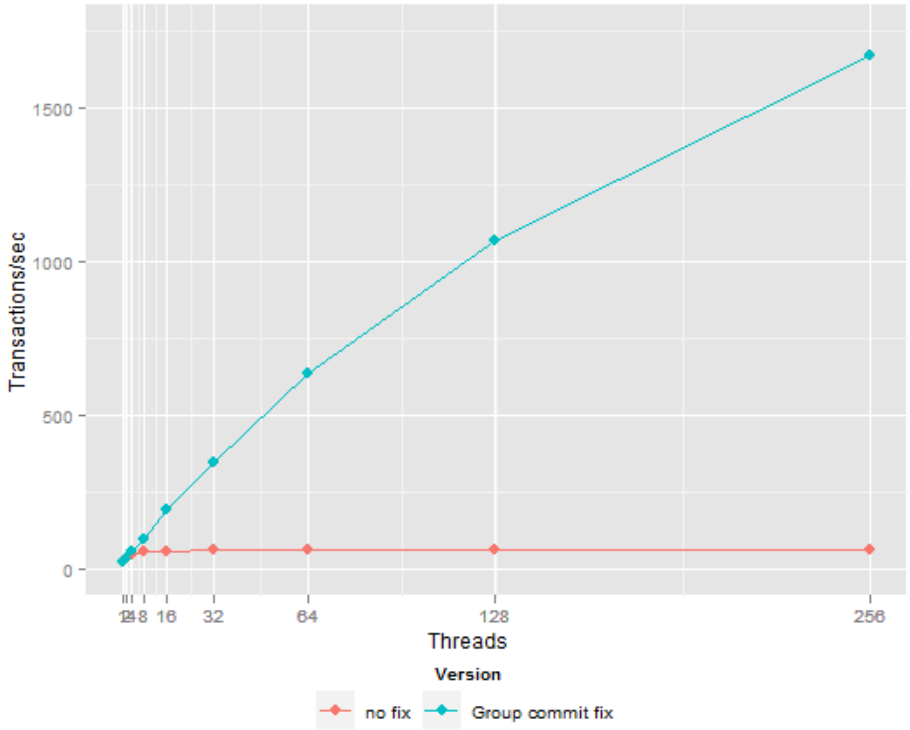
- Есть крутая теорема о сериализации транзакций
- Читаем про неё у Вани: http://plumqqz.livejournal.com/387380.html
- Коротко говоря: можно, но есть нюансы (в них дьявол)
- PostgreSQL — **IO-bound**, диск не параллелится, **хорошо**
- MySQL — **CPU-bound**, параллелить нельзя, **плохо**
- Мощный сервер о 12 ядрах, занято одно ядро
- Лимитирует **скорость MySQL** репликация **тактовая частота ядра, а не количество ядер**

Давай сгруппируем

- Транзакции можно группировать
- MySQL: innodb_flush_log_at_trx_commit = 0 либо innodb_flush_log_at_trx_commit = 2
- Однако, MySQL нужно писать в два журнала
- Group Binary Log Commit — 5.6 / 5.7
- Два журнала — нужен two-phase-commit

  http://en.wikipedia.org/wiki/Two-phase_commit_protocol

Group Binary Log Commit



http://www.percona.com/blog/2011/07/13/testing-the-group-commit-fix/

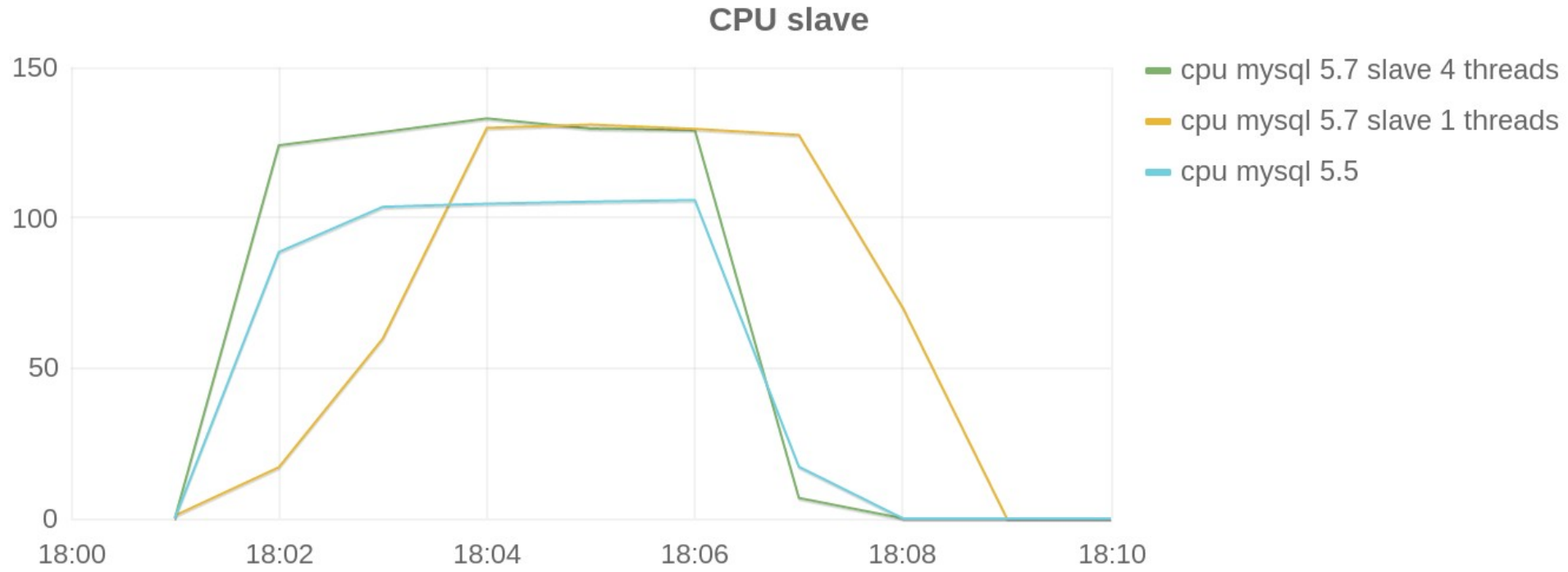
Группы и параллелизм

- Сгруппированные коммиты можно применять параллельно
- MySQL 5.6: DB, MySQL 5.7: DB и LOGICAL_CLOCK
- Если на master много взаимонепересекающихся коммитов, они попадут в одну группу
- Ниже опции которые нужно крутить
- **Master**: binlog_order_commits, binlog_max_flush_queue_time, innodb_flush_log_at_trx_commit
- **Slave**: slave_parallel_type, slave_parallel_workers

Давайте измерим

- MySQL 5.5.40
- MySQL 5.7.5 m15
- **statement-based replication** — нам так нужно
- Mail.Ru Target — на row-based **десятки** терабайт логов в сутки
- sysbench, 16 потоков, oltp-complex, таблица 10^6 строк
- Intel(R) Xeon(R) CPU E5-2420 0 @ 1.90GHz
- Master & Slave — 4 cores, Client — 16 cores
- https://github.com/zamotivator/2014-highload-mysql

Результаты



Анализ результатов

- Один поток 5.7 работает **хуже** 5.5
- Четыре потока 5.7 работают **примерно** как 5.5
- Для параллелизма нужен group binary log commit на master (5.7)
- **Миграция без downtime** master 5.5, slave 5.7 **будет отставать**

Выводы

Выводы: типы журналов

- Физическая репликация — **уровень хранения**
- Логическая репликация — **уровень данных**
- ??? — **уровень запросов**
- **Уровень хранения**: InnoDB Undo/Redo, PostgreSQL WAL
- **Уровень данных**: MySQL row-based
- **Уровень запросов**: MySQL statement-based
- У каждого решения свои плюсы и минусы

Выводы: Master

- **PostgreSQL** репликация не оказывает* влияния на master
- **MySQL statement-based** репликация требует жертв (innodb_locks_unsafe_for_binlog)
- **MySQL row-based** репликация пишет на диск в полтора раза больше PostgreSQL
- **MySQL** репликация без group binary log commit убивает линейное масштабирование

Выводы: Slave

- **PostgreSQL** репликация как правило **IO-bound**
- **MySQL** репликация как правило **CPU-bound**

Выводы: Slave IO

- **PostgreSQL** репликация генерирует **много*** бинарных логов
- **MySQL row-based** репликация генерирует **много*** бинарных логов
- **MySQL statement-based** репликация генерирует **мало*** логов

Выводы: Slave CPU

- **PostgreSQL** репликация как правило **IO-bound**
- **MySQL** репликация требует хороших* индексов на slave
- **MySQL statement-based** репликация работает как однопоточный master
- **MySQL parallel slave** выглядит многообещающим, но не работает*

Выводы: Consistentcy

- **PostgreSQL** репликация работает **корректно всегда**
- **MySQL row-based репликация** работает **корректно для DML**
- **MySQL row-based репликация** может работать **некорректно для DDL** (потому что DDL идёт как statement-based)
- **MySQL statement-based** репликация может работать **некорректно для DML и DDL**
- **Mixed-based** репликация — как повезёт

Выводы: Flexiblity

- **PostgreSQL** реплика бинарная копия мастера
- **MySQL** репликация более гибка:
  - иная схема
  - другие индексы
- **MySQL** репликация имеет libslave
- Однако:
  - **PostgreSQL Logical Log Streaming Replication** https://wiki.postgresql.org/wiki/Logical_Log_Streaming_Replication
  - **PostgreSQL Logical Decoding** http://www.postgresql.org/docs/9.4/static/logicaldecoding.html

И напоследок

- Репликация это не бекапы
- Таблица — гомогенное мультимножество кортежей

Благодарности

- Константин Осипов, Mail.Ru Group, MySQL, Tarantool — критика
- Дмитрий Ленев, Oracle — критика и помощь
- Александр Горный, Mail.Ru Group — критика, вычистка, помощь
- Александр Чистяков, Git in Sky — бенчмарки, вопросы
- Максим Оранский — вычитка и советы
- Федор Сигаев, Mail.Ru Group, PostgreSQL — советы, вопросы
- Дмитрий Кирноценский, Mail.Ru Group — разрешение опубликовать данные
- Дмитрий Школьников, Mail.Ru Group — оформление

Контакты

- Личный: oleg@oleg.sh
- Рабочий: o.tsarev@corp.mail.ru
- Facebook: https://www.facebook.com/oleg.i.tsarev
- Github: https://github.com/zamotivator