OpenSource SQL Databases Enter Millions Queries per Second Era

Анастасия Распопина, Света Смирнова, Александр Коротков, Фёдор Сигаев

8 ноября 2016

HighLoad++

Профессиональная конференция разработчиков высоконагруженных систем

Света Смирнова

- Инженер тех. поддержки MySQL
- Автор
  - MySQL Troubleshooting
  - JSON UDF функции
  - FILTER clause для MySQL
- Докладчик
  - Percona Live, OOW, Fosdem, DevConf, ...

Russian developers of PostgreSQL:

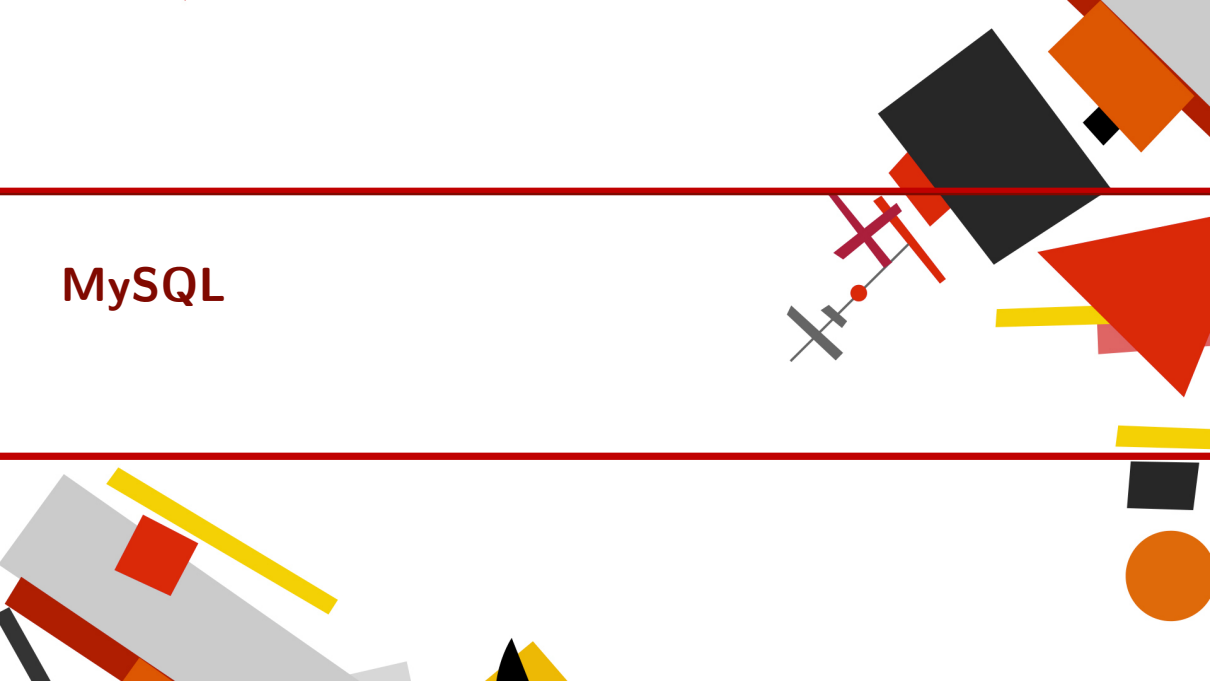
Alexander Korotkov, Teodor Sigaev, Oleg Bartunov

- Speakers at PGCon, PGConf: 20+ talks
- GSoC mentors
- 3 PostgreSQL major contributors + 1 committer
- Conference organizers
- 50+ years of PostgreSQL expertship: dev., audit, consult.
- Postgres Professional company co-founders

- **PostgreSQL CORE**
- Locale support
- PostgreSQL extendability:
- GiST(KNN), GIN, SP-GiST
- Full Text Search (FTS)
- NoSQL (hstore, jsonb)
- Indexed regexp search
- Access method extendability

- **Extensions**
- intarray
- pg_trgm
- ltree
- hstore
- plantuner
- jsquery

MySQL



Со стороны MySQL

- Для меня всё могло быть просто

Со стороны MySQL

- Для меня всё могло быть просто
  - Дмитрий регулярно публикует детальные результаты тестов

Со стороны MySQL

- Для меня всё могло быть просто
  - Дмитрий регулярно публикует детальные результаты тестов
  - Александр мог прогнать их на PostgreSQL

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Многие клиенты используют несколько БД
  - Как правило хорошо знают только одну
  - Общие проблемы

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Многие клиенты используют несколько БД
  - Как правило хорошо знают только одну
  - Общие проблемы
    - Скорость записи на мастере

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Многие клиенты используют несколько БД
  - Как правило хорошо знают только одну
  - Общие проблемы
    - Скорость записи на мастере
    - Максимальная производительность read-only slave

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Многие клиенты используют несколько БД
  - Как правило хорошо знают только одну
  - Общие проблемы
    - Скорость записи на мастере
    - Максимальная производительность read-only slave
    - Checksums, синхронизация, компрессия

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Многие клиенты используют несколько БД
  - Как правило хорошо знают только одну
  - Общие проблемы
  - Один вопрос: как получить лучшие результаты для каждой из баз на одинаковом оборудовании?

Со стороны MySQL

- Для меня всё могло быть просто
- Оригинальная цель исследования
  - Один вопрос: как получить лучшие результаты для каждой из баз на одинаковом оборудовании?
- Нужно использовать одинаковые тесты

Трудности, с которыми я столкнулась

Неожиданности read-write

- Машина Percona
  - Процессоры: physical = 2, cores = 12, virtual = 24, hyperthreading = yes
  - Память: 251.9G
  - Скорость диска: about 33K IOPS
- Машина Freematiq
  - Процессоры: physical = 4, cores = 72, virtual = 144, hyperthreading = yes
  - Память: 3,0T
  - Скорость диска: about 3K IOPS

Первые результаты

- Машина Percona

```
OLTP test statistics:
 transactions:                          1000000 (28727.81 per sec.)
 read/write requests:                   5000000 (143639.05 per sec.)
 other operations:                      2000000 (57455.62 per sec.)
```

- Машина Freematiq

```
 transactions:                          1000000 (29784.74 per sec.)
 read/write requests:                   5000000 (148923.71 per sec.)
 other operations:                      2000000 (59569.49 per sec.)
```

- Производительность одинаковая
- Я бы хотела использовать все ядра!

Попытка #2: Read-only 100% в памяти

- Сразу получилось 700 QPS
- SysBench использовал столько же CPU, сколько и MySQL

```
PID  USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
4585 smirnova  20   0  0,157t 0,041t   9596 S  7226  1,4  12:27.16 mysqld
8745 smirnova  20   0 1266212 629148   1824 S  7126  0,0   9:22.78 sysbench
```

Попытка #2: Read-only 100% в памяти

- Сразу получилось 700 QPS
- SysBench использовал столько же CPU, сколько и MySQL
- Решение
  - Запускать sysbench с опциями –percentile=0 –max-requests=0
  - Ветка concurrency_kit в SysBench
  - Переписать *lua скрипты с использованием Prepared Statements

Попытка #2: Read-only 100% в памяти

- Сразу получилось 700 QPS
- SysBench использовал столько же CPU, сколько и MySQL
- Решение
- Написать хорошие тесты - это непросто!

Промежуточные результаты и аномалии

Системная конфигурация

- vm.swappiness=1
- cpupower frequency-set –governor performance
- kernel.sched_autogroup_enabled=0
- kernel.sched_migration_cost_ns= 5000000
- vm.dirty_background_bytes=67108864
- vm.dirty_bytes=536870912
- IO scheduler [deadline]

RO конфигурация MySQL

- 5.7.15-9 Percona Server (GPL), Release 9
- За основу взята конфигурация Дмитрия
- (С небольшими изменениями)

RO конфигурация MySQL

- 5.7.15-9 Percona Server (GPL), Release 9
- За основу взята конфигурация Дмитрия
- General-Files

```
# general
  table_open_cache = 8000
  table_open_cache_instances=16
  back_log=1500
  query_cache_type=0
  max_connections=4000

# files
  innodb_file_per_table
  innodb_log_file_size=1024M
  innodb_log_files_in_group=3
  innodb_open_files=4000
```

RO конфигурация MySQL

- 5.7.15-9 Percona Server (GPL), Release 9
- За основу взята конфигурация Дмитрия
- Мониторинг-Buffers

```
# Monitoring
  innodb_monitor_enable = '%'
  performance_schema=OFF #cpu-bound, matters for performance

#Percona Server specific
  userstat=0
  thread-statistics=0

# buffers
  innodb_buffer_pool_size= 128000M (32000M)
  innodb_buffer_pool_instances=128 (32)
  innodb_log_buffer_size=64M
```

RO конфигурация MySQL

- 5.7.15-9 Percona Server (GPL), Release 9
- За основу взята конфигурация Дмитрия
- InnoDB-специфичные

```
# tune
  innodb_checksums=1 (0)
  innodb_doublewrite= 1
  innodb_support_xa=0
  innodb_thread_concurrency=0
  innodb_flush_log_at_trx_commit=2
  innodb_flush_method=O_DIRECT_NO_FSYNC
  innodb_max_dirty_pages_pct=90
  innodb_max_dirty_pages_pct_lwm=10
  innodb_lru_scan_depth=4000
  innodb_page_cleaners=4
  innodb_use_native_aio=1
  innodb_stats_persistent = 1
  innodb_spin_wait_delay=6
  join_buffer_size=32K
  sort_buffer_size=32K
```

RO конфигурация MySQL

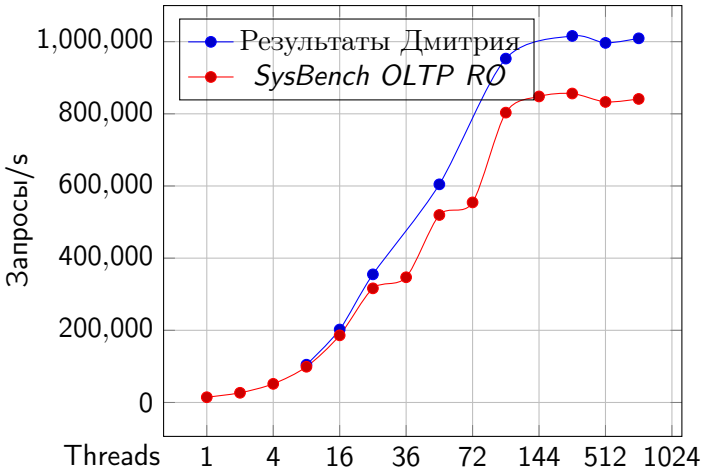
- 5.7.15-9 Percona Server (GPL), Release 9
- За основу взята конфигурация Дмитрия
- Для производительности

```
# perf special
  innodb_adaptive_flushing = 1
  innodb_flush_neighbors = 0
  innodb_read_io_threads = 4
  innodb_write_io_threads = 4
  innodb_io_capacity=2000
  innodb_io_capacity_max=4000
  innodb_purge_threads=4
  innodb_max_purge_lag_delay=30000000
  innodb_max_purge_lag=0
  innodb_adaptive_hash_index=0
```

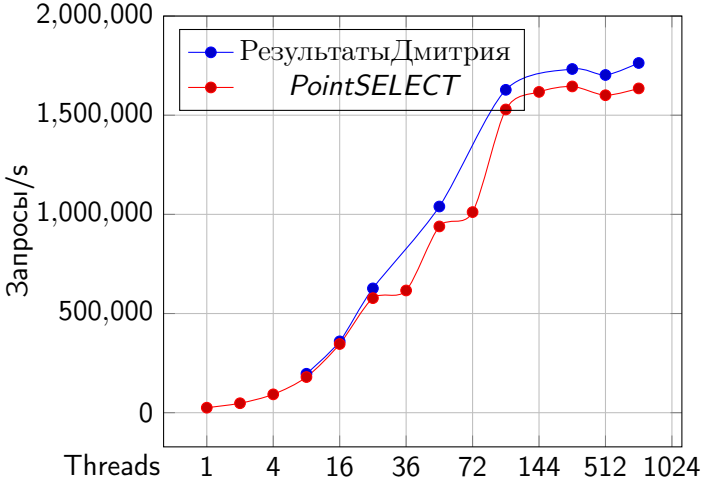
Параметры sysbench

```
LD_PRELOAD=/data/sveta/5.7.14/lib/mysql/libjemalloc.so \
/data/sveta/sbkk/bin/sysbench \
[-test=/data/sveta/sysbench/sysbench/tests/db/oltp_prepared.lua |\
-test=/data/sveta/sysbench/sysbench/tests/db/oltp_simple_prepared.lua \]
-db-driver=mysql -oltp-tables-count=8 -oltp-table-size=10000000 \
-mysql-table-engine=innodb -mysql-user=msandbox \
-mysql-password=msandbox -mysql-socket=/tmp/mysql_sandbox5715.sock \
-num-threads=$i -max-requests=0 -max-time=300 -percentile=0 \
[-oltp-read-only=on -oltp-skip-trx=on \]
run
```

Результаты RO хуже, чем у Дмитрия



Результаты RO хуже, чем у Дмитрия



RW конфигурация MySQL

- Open files - Monitoring

```
#Open files
table_open_cache = 8000
table_open_cache_instances = 16
query_cache_type = 0
join_buffer_size=32k
sort_buffer_size=32k
max_connections=16000
back_log=5000
innodb_open_files=4000

#Monitoring
performance-schema=0

#Percona Server specific
userstat=0
thread-statistics=0
```

RW конфигурация MySQL

- InnoDB general и concurrency

```
#General
innodb_buffer_pool_load_at_startup=1
innodb_buffer_pool_dump_at_shutdown=1
innodb_numa_interleave=1
innodb_file_per_table=1
innodb_file_format=barracuda
innodb_flush_method=O_DIRECT_NO_FSYNC
innodb_doublewrite=1
innodb_support_xa=1
innodb_checksums=1

#Concurrency
innodb_thread_concurrency=144
innodb_page_cleaners=8
innodb_purge_threads=4 #with 1 seem to be more stable
innodb_spin_wait_delay=12 #Good value for RO is 6, for RW and RC is 192
```

RW конфигурация MySQL

- InnoDB buffer, log и IO capacity

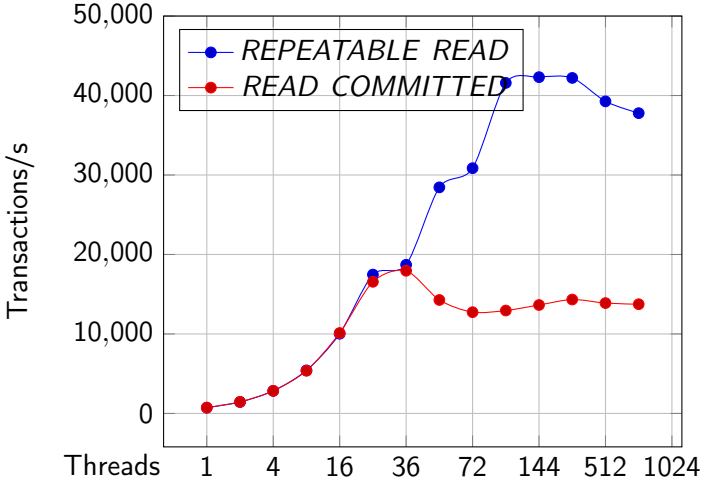
```
innodb_log_file_size=8G
innodb_log_files_in_group=16
innodb_buffer_pool_size=128G
innodb_buffer_pool_instances=128
innodb_io_capacity=18000
innodb_io_capacity_max=36000
innodb_flush_log_at_timeout=0
innodb_flush_log_at_trx_commit=2
```

RW конфигурация MySQL

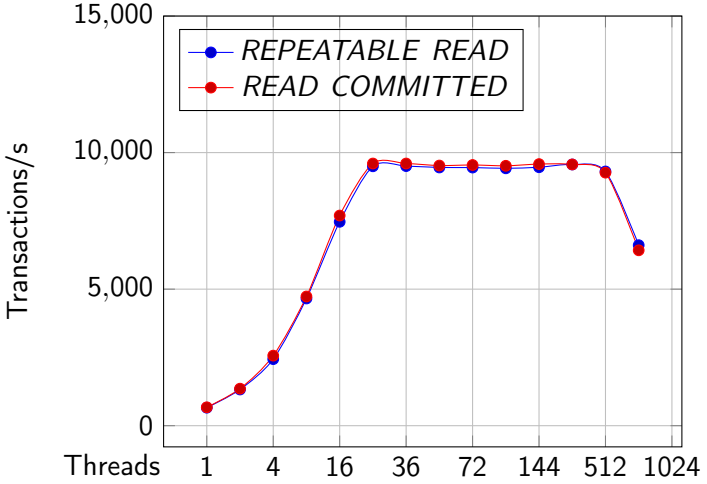
- InnoDB performance, load specific

```
innodb_flush_sync=1
innodb_adaptive_flushing=1
innodb_flush_neighbors = 0
innodb_max_dirty_pages_pct=90
innodb_max_dirty_pages_pct_lwm=10
innodb_lru_scan_depth=4000
innodb_adaptive_hash_index=0
innodb_change_buffering=none #can be inserts
optimizer_switch="index_condition_pushdown=off"
```

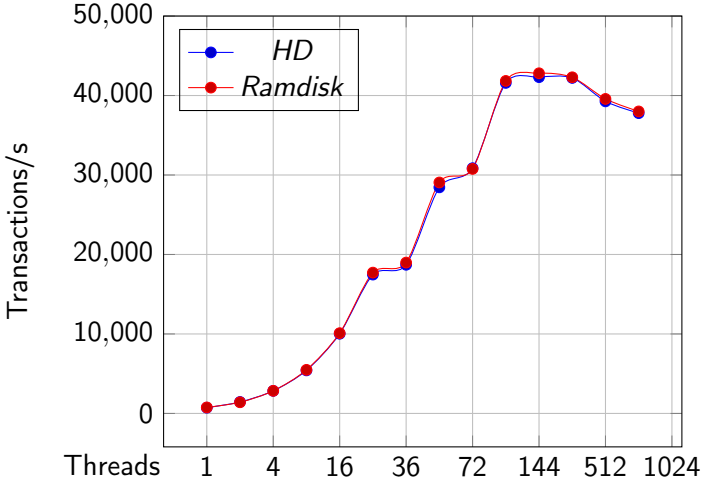
Аномалия READ COMMITTED



RR vs RC на 24 ядрах

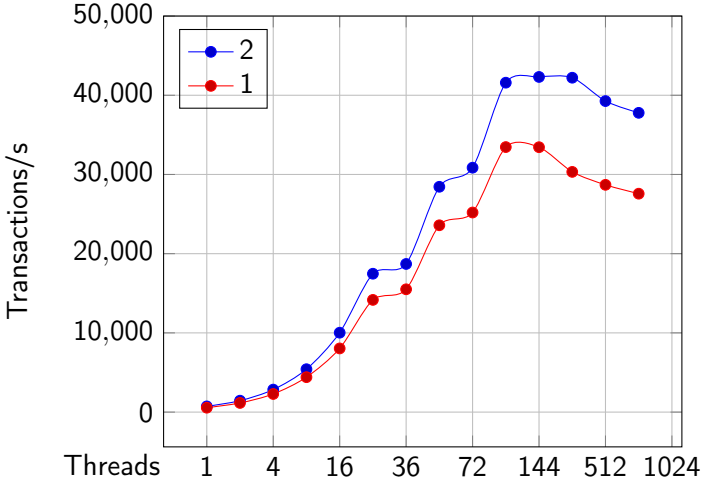


Аномалия IO

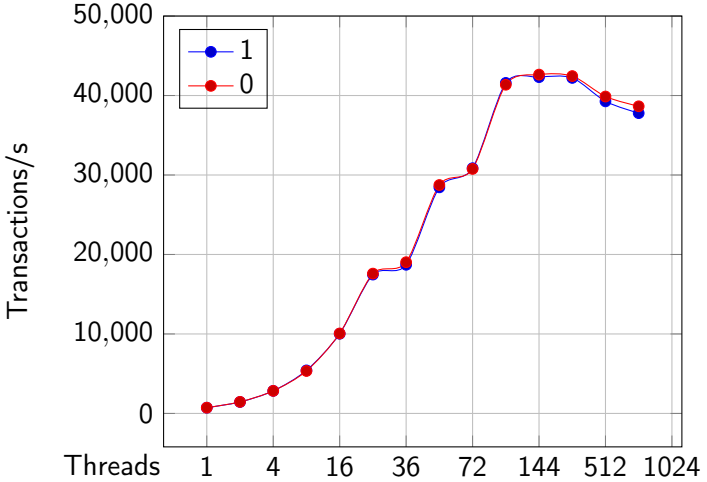


Полезные сравнения

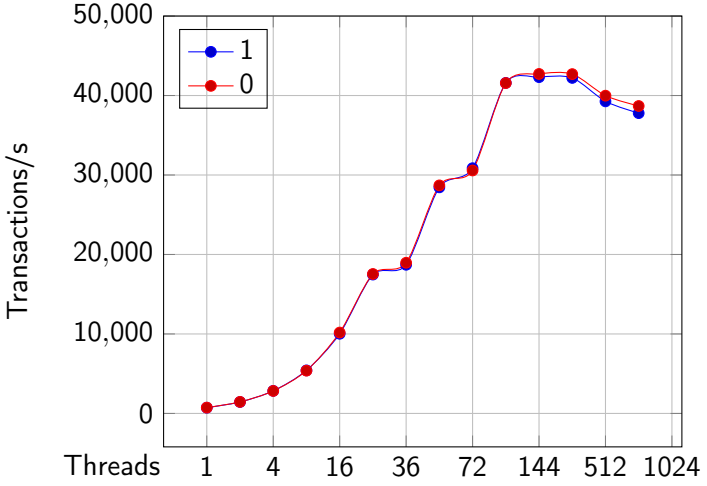
innodb_flush_log_at_trx_commit 1 vs 2



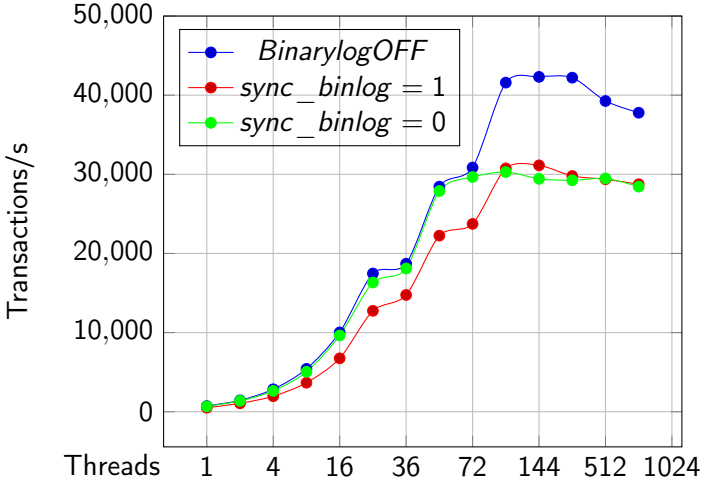
innodb_doublewrite 1 vs 0



innodb_checksums 1 vs 0



sync_binlog 1 vs 0



MySQL: почему эти результаты стали возможны

Ключевые изменения

- InnoDB: оптимизация transaction list

Ключевые изменения

- InnoDB: оптимизация transaction list
  - Версия 5.7.2: глобальный transaction был разбит на два
    - Read-write
    - Read-only

Ключевые изменения

- InnoDB: оптимизация transaction list
  - Версия 5.7.2: глобальный transaction был разбит на два
    - Read-write
    - Read-only
  - Версия 5.7.3: по умолчанию транзакция независима, если только не была открыта с опцией READ WRITE
    - Read only транзакции mutex-free
    - READ ONLY транзакций нет в выводе SHOW ENGINE INNODB STATUS

Ключевые изменения

- InnoDB: оптимизация transaction list
  - Версия 5.7.2: глобальный transaction был разбит на два
    - Read-write
    - Read-only
  - Версия 5.7.3: по умолчанию транзакция независима, если только не была открыта с опцией READ WRITE
    - Read only транзакции mutex-free
    - READ ONLY транзакций нет в выводе SHOW ENGINE INNODB STATUS
  - Подробнее
  - WL #6047

Ключевые изменения

- InnoDB: оптимизация transaction list
- InnoDB: уменьшен lock_sys_t::mutex contention, WL #6899

Ключевые изменения

- InnoDB: оптимизация transaction list
- InnoDB: уменьшен lock_sys_t::mutex contention, WL #6899
- InnoDB: исправлен index->lock contention WL #6326

Ключевые изменения

- InnoDB: быстрый и параллельный flushing

Ключевые изменения

- InnoDB: быстрый и параллельный flushing
  - Несколько потоков page cleaner: WL #6642

Ключевые изменения

- InnoDB: быстрый и параллельный flushing
  - Несколько потоков page cleaner: WL #6642
  - Улучшен адаптивный flushing: WL #7868

Ключевые изменения

- InnoDB: быстрый и параллельный flushing
  - Несколько потоков page cleaner: WL #6642
  - Улучшен адаптивный flushing: WL #7868
  - Уменьшено число страниц, которое должно быть flushed: WL #7047

Ключевые изменения

- Масштабируемость MDL (Meta-Data Lock)

Ключевые изменения

- Масштабируемость MDL (Meta-Data Lock)
  - Убран THR_LOCK::mutex для InnoDB: WL #6671

Ключевые изменения

- Масштабируемость MDL (Meta-Data Lock)
  - Убран THR_LOCK::mutex для InnoDB: WL #6671
  - Разбит на части LOCK_grant
    - Количество постоянное
    - Thread ID используется для выбора партиции
    - WL #8355
    - Bug #72829

Ключевые изменения

- Масштабируемость MDL (Meta-Data Lock)
  - Убран THR_LOCK::mutex для InnoDB: WL #6671
  - Разбит на части LOCK_grant
  - Lock-free MDL lock acquisition для DML: WL #7306, WL #7305

Другие улучшения производительности MySQL

- Performance Schema быстрее, чем раньше
  - Я заметила влияние не на всех тестах
  - Я НЕ включала никаких инструментов

Другие улучшения производительности MySQL

- Performance Schema быстрее, чем раньше
- Быстрый innodb_checksum_algorithm crc32 по умолчанию

Другие улучшения производительности MySQL

- Performance Schema быстрее, чем раньше
- Быстрый innodb_checksum_algorithm crc32 по умолчанию
- Производительность InnoDB Temporary Table
  - Нет UNDO и REDO logging
  - Нет Insert buffering
  - Нет persistence
  - WL #6469, WL #6470, WL #6915, https://goo.gl/LeIYD4

Другие улучшения производительности MySQL

- Performance Schema быстрее, чем раньше
- Быстрый innodb_checksum_algorithm crc32 по умолчанию
- Производительность InnoDB Temporary Table
- Выгрузка и загрузка InnoDB buffer pool

Другие улучшения производительности MySQL

- Performance Schema быстрее, чем раньше
- Быстрый innodb_checksum_algorithm crc32 по умолчанию
- Производительность InnoDB Temporary Table
- Выгрузка и загрузка InnoDB buffer pool
- И много чего ещё ☺

PostgreSQL

Трудности

sysbench и prepared statements: попытка 1

- Проблема: обработка NULL для PostgreSQL сломана в sysbench.

```
[fontsize=]
FATAL: failed to execute function ‘event’: 3
(last message repeated 7 times)
FATAL: PQexecPrepared() failed: 7 ERROR:  invalid input syntax for integer:
```

- Починено. Алексей Копытов смёржил pull request.

```
[fontsize=]
     /* Convert SysBench bind structures to PgSQL data */
     for (i = 0; i < (unsigned)pgstmt->nparams; i++)
     {
-      if (stmt->bound_param[i].is_null)
+      if (stmt->bound_param[i].is_null && *(stmt->bound_param[i].is_null))
         continue;

       switch (stmt->bound_param[i].type) {
```

sysbench и prepared statements: попытка 2

- Проблема 2: sysbench не может нагрузить PostgreSQL, когда используются prepared statements.

```
[fontsize=]
 93087 korotkov  20   0 9289440 3,718g   2964 S 242,6  0,1   0:32.82 sysbench
 93161 korotkov  20   0 32,904g  81612  80208 S   4,0  0,0   0:00.47 postgres
 93116 korotkov  20   0 32,904g  80828  79424 S   3,6  0,0   0:00.46 postgres
 93118 korotkov  20   0 32,904g  80424  79020 S   3,6  0,0   0:00.47 postgres
 93121 korotkov  20   0 32,904g  80720  79312 S   3,6  0,0   0:00.47 postgres
 93128 korotkov  20   0 32,904g  77936  76536 S   3,6  0,0   0:00.46 postgres
 93130 korotkov  20   0 32,904g  81604  80204 S   3,6  0,0   0:00.47 postgres
 93146 korotkov  20   0 32,904g  81112  79704 S   3,6  0,0   0:00.46 postgres
.....................................................................
```

- ...решено пока забить на sysbench и пользоваться pgbench!

OLTP RO скрипт для pgbench

```
\set table_size 10000000
\set range_size 100
\set id1 random(1, :table_size)
.....................................................
\set id10 random(1, :table_size)
\set r1l random(1, :table_size)
\set r1u :r1l + :range_size
.....................................................
\set r4l random(1, :table_size)
\set r4u :r4l + :range_size
SELECT c FROM sbtest WHERE id = :id1;
.....................................................
SELECT c FROM sbtest WHERE id = :id10;
SELECT c FROM sbtest WHERE id BETWEEN :r1l AND :r1u;
SELECT SUM(K) FROM sbtest WHERE id BETWEEN :r2l AND :r2u;
SELECT c FROM sbtest WHERE id BETWEEN :r3l AND :r3u ORDER BY c;
SELECT DISTINCT c FROM sbtest WHERE id BETWEEN :r4l AND :r4u;
```

OLTP RW скрипт для pgbench

```
\set table_size 10000000
.....................................................
\set u1 random(1, :table_size)
\set u2 random(1, :table_size)
\set u3 random(1, :table_size)
\set u4 random(1, :table_size)
BEGIN;
SELECT c FROM sbtest WHERE id = :id1;
.....................................................
SELECT DISTINCT c FROM sbtest WHERE id BETWEEN :r4l AND :r4u;
UPDATE sbtest SET k = k + 1 WHERE id = :u1;
UPDATE sbtest SET c = sb_rand_str('###########-###########-###########-###########-###########-########
DELETE FROM sbtest WHERE id = :u3;
INSERT INTO sbtest (id, k, c, pad) VALUES (:u3, :u4, sb_rand_str('###########-###########-###########-#
COMMIT;
```

Пришлось реализовать sb_rand_str() на стороне PostgreSQL на C.

Как его запускать?

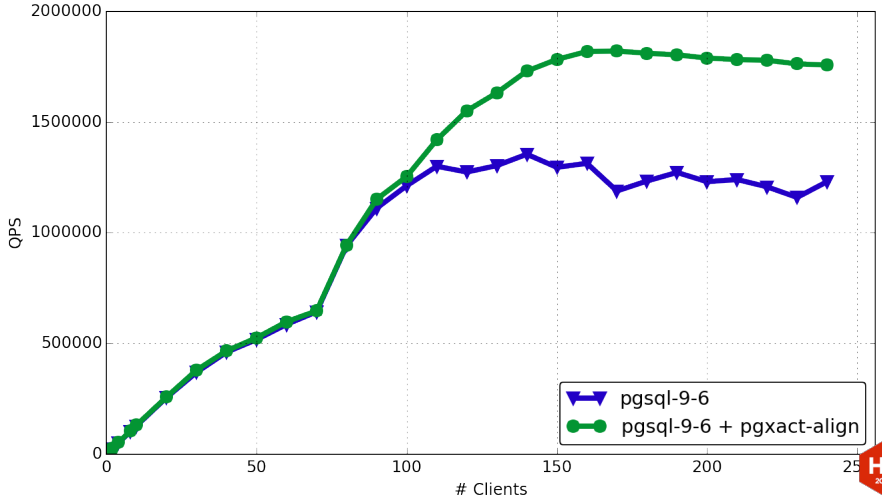
Всё на github'е и воспроизводится!

```
[fontsize=]
$ git clone https://github.com/postgrespro/pg_oltp_bench.git
$ cd pg_oltp_bench
$ make USE_PGXS=1
$ sudo make USE_PGXS=1 install
$ psql DB -f oltp_init.sql
$ psql DB -c "CREATE EXTENSION pg_oltp_bench;"
$ pgbench -c 100 -j 100 -M prepared -f oltp_ro.sql -T 300 -P 1 DB
$ pgbench -c 100 -j 100 -M prepared -f oltp_rw.sql -T 300 -P 1 DB
```

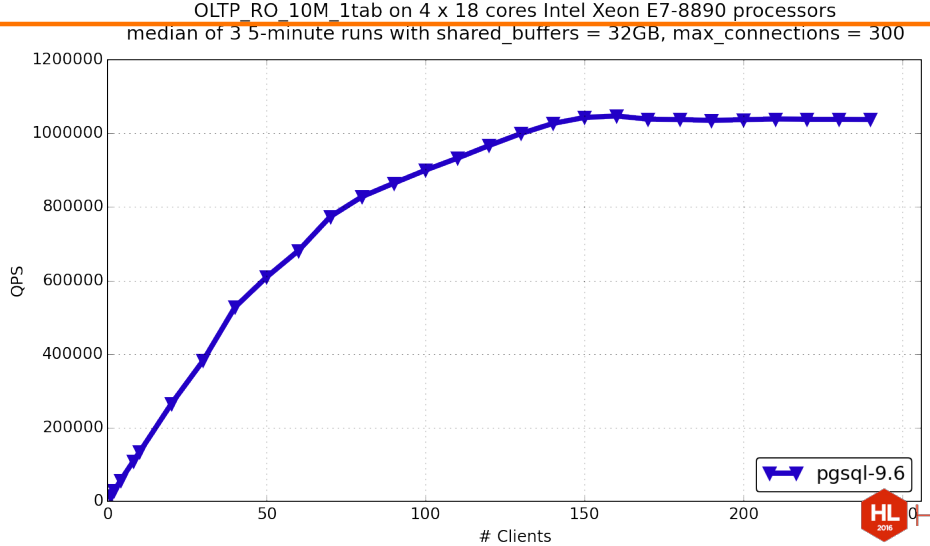
Результаты

Результаты: точечные запросы

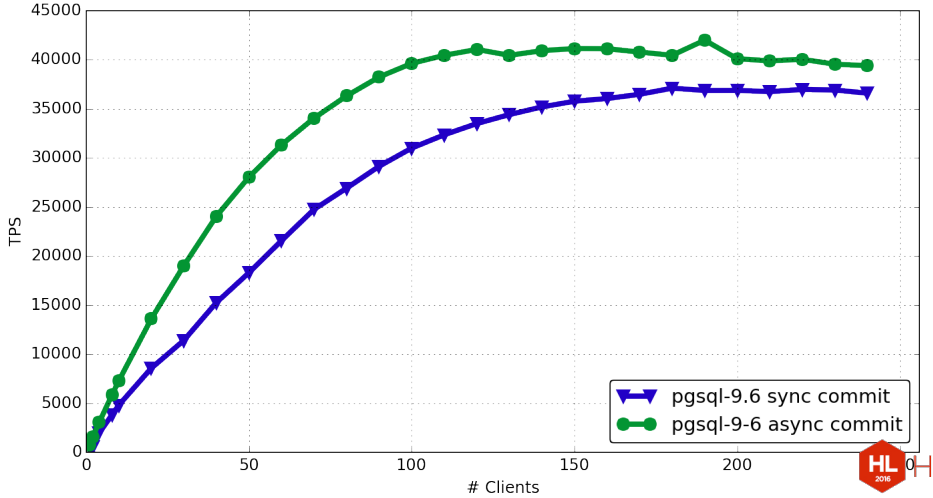
pgbench -s 1000 -j $n -c $n -M prepared -S on 4 x 18 cores Intel Xeon E7-8890 processors
median of 3 5-minute runs with shared_buffers = 32GB, max_connections = 300



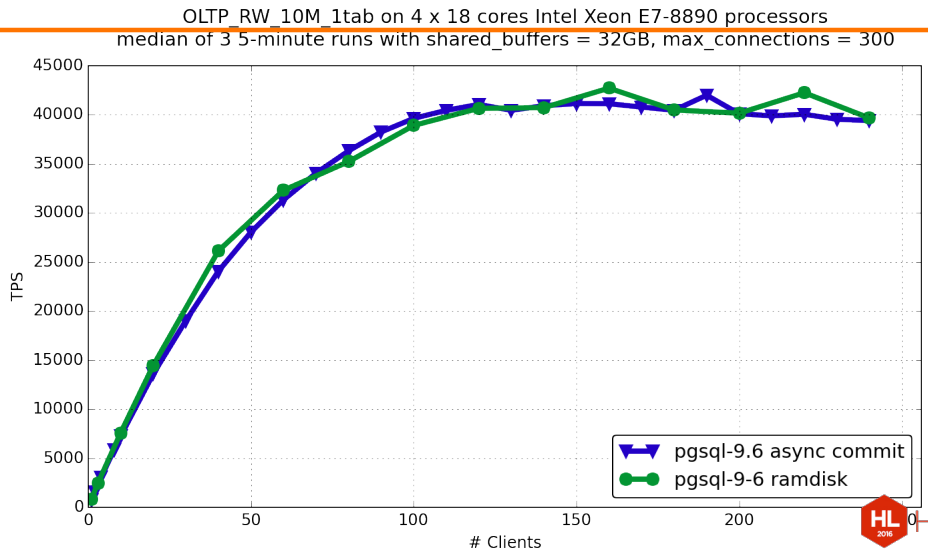
Результаты: OLTP RO



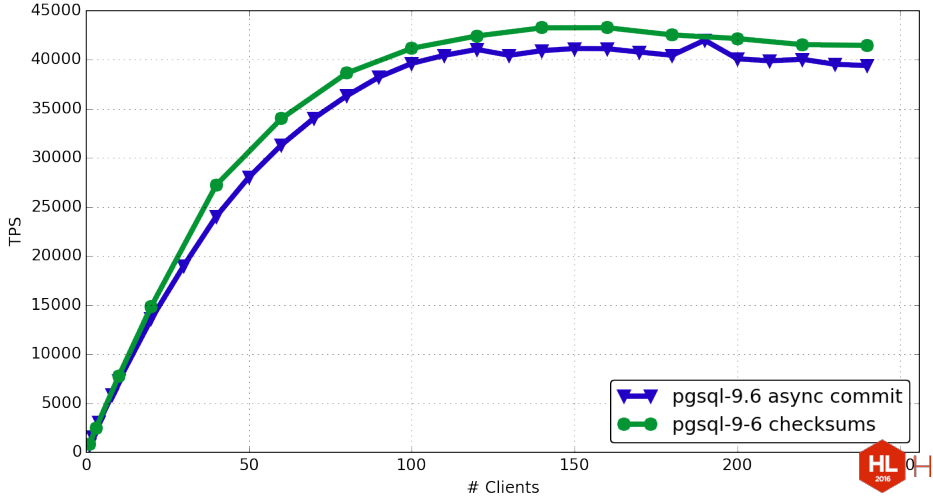
Результаты: OLTP RW



Результаты: OLTP RW ramdisk



Результаты: OLTP RW checksums



Как это стало возможно

Pin/UnpinBuffer in lockless manner

Перед тем как “потрогать” любой блок данных, backend вначале должен “запинить” соответствующий буфер. Pin/UnpinBuffer – очень частая операция.

Было:

```
S_LOCK(bufHdr);
bufHdr->pinCount++;
S_UNLOCK(bufHdr);
```

Стало:

```
atomic_increment(buf_hdr->pinCount);
```

См. commit: https://goo.gl/LLCvR8.

Уменьшение конкуренции за ProcArrayLock

- Снапшот содержит список id запущенных транзакций. Для получения снапшота нужно взять shared ProcArrayLock.
- Commit транзакции очищает её id из разделяемой памяти. Для commit'а транзакции нужен exclusive ProcArrayLock.
- Высокий TPS приводит к высокой конкуренции за ProcArrayLock.
- Решение: очищать id транзакций пачками.

См. commit: https://goo.gl/ZxiilI.

Уменьшение конкуренции за CLogControlLock

- Получение статуса транзакции требует shared CLogControlLock. Установка статуса транзакции требует exclusive CLogControlLock. Чтение станицы CLOG требует exclusive CLogControlLock.
- На современных многоядерных архитектурах, backend'ы часто получают статусы транзакций. Число запрашиваемых транзакций также высоко.
- Решение: увеличить число буферов CLOG с 32 до 128. Благодаря этого страницы CLOG нужно будет реже читать.

See commit details: `https://goo.gl/aaPYsJ`.

Перспективы

Где бутылочное горлышко у PostgreSQL?

Где бутылочные горлышки у PostgreSQL?

Где бутылочные горлышки у PostgreSQL?

- Buffer manager – медленная хэш таблица, “пин”, блокировки.
- Снэпшоты – для получения каждого снэпшота нужно просканировать все активные транзакции.
- Синхронный протокол.
- Медленное выделение id транзакции – много блокировок.

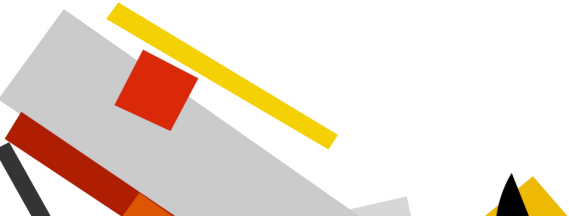
Бутылочные горлышки PostgreSQL в цифрах

- SELECT val FROM tab WHERE id IN (:id1, ... :id10) – 150K в секунду = 1.5M точечных запросов в секунду, нет выигрыша. Упираемся в buffer manager.
- 10 x SELECT 1 в одну команду – 2.2M запросов в секунду. Упираемся во взятие снэпшота.
- SELECT 1 с CSN патчем (дешёвые снэпшоты) – 3.9M запросов в секунду. Упираемся в протокол.
- SELECT txid_current() – 390K запросов в секунду. Упираемся в блокировки.

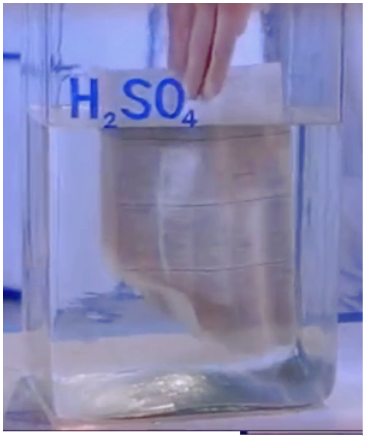
Как улучшить PostgreSQL?

- In-memory табличный движок без buffer manager'а.
- CSN для быстрого взятие снапшотов.
- Асинхронный бинарный протокол позволит обработать больше коротких запросов.
- Lockless выделение id транзакций.

Сравнение

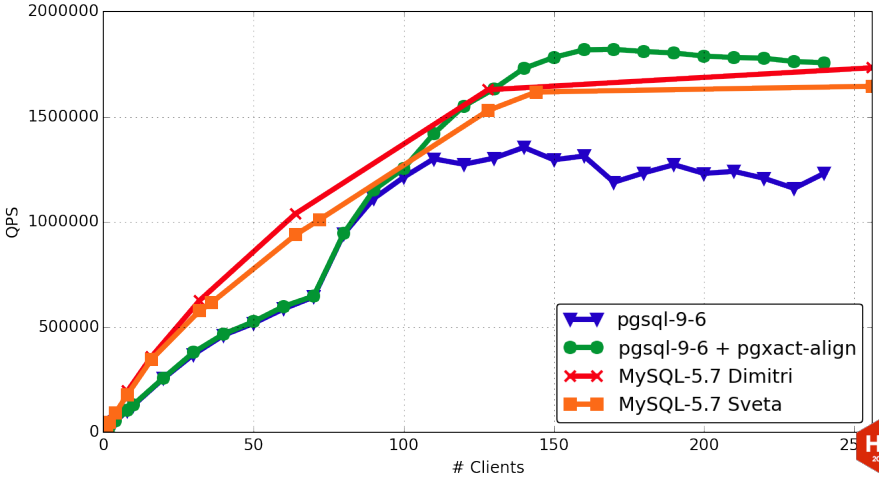


Неравное сравнение!

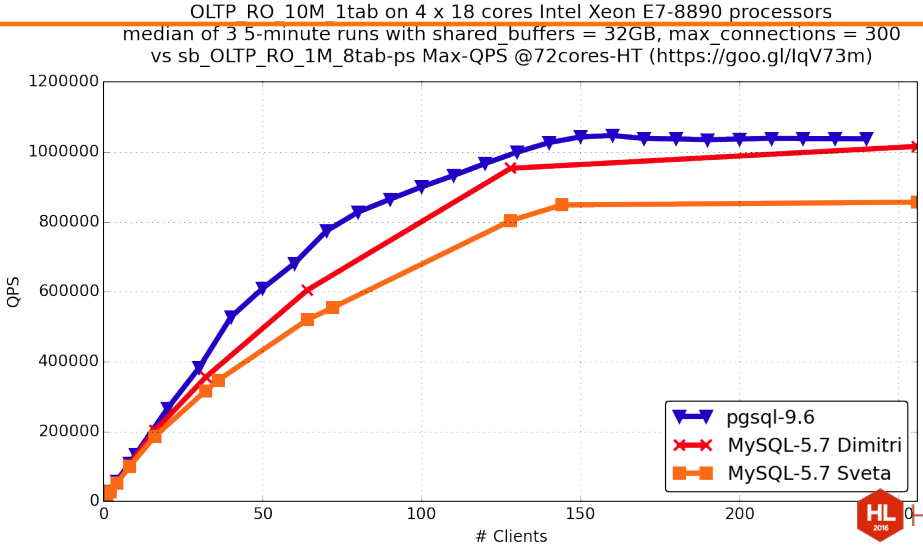


Сравнение: точечные запросы

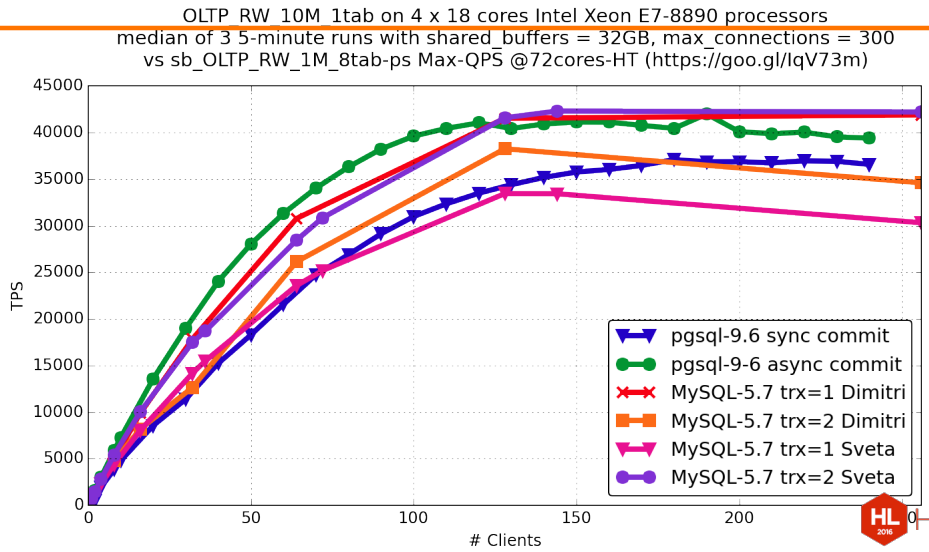
pgbench -s 1000 -j $n -c $n -M prepared -S on 4 x 18 cores Intel Xeon E7-8890 processors
median of 3 5-minute runs with shared_buffers = 32GB, max_connections = 300
vs sb_RO_Pselects_1M_8tab-ps-socket-dim Max-QPS @72cores-HT



Сравнение: OLTP RO



Сравнение: OLTP RW



Было близко…

Итоги

Итоги

- MySQL и PostgreSQL – динамично развивающиеся open source СУБД, быстро адаптирующиеся под современной железо.
- Релизы MySQL 5.7 и PostgreSQL 9.6 содержат серьёзные улучшения вертикальной масштабируемости (consider upgrade).

Благодарности

- Freematiq за предоставленные сервера
- MySQL Server Team
- MySQL InnoDB Team
- Dimitri Kravtchuk
- Alexey Kopytov

Использованные инструменты

- sysbench
- pgbench
- pg_oltp_bench
- open-database-bench

У вас есть вопросов,

у нас есть ответов.

Спасибо!

http://www.slideshare.net/SvetaSmirnova

https://twitter.com/svetsmirnova

https://github.com/svetasmirnova

http://www.postgrespro.ru

https://github.com/postgrespro