# Рецепты оптимизации производительности PostgreSQL

Стачка
2016

Алексей Ермаков
alexey.ermakov@postgresql-consulting.com

PostgreSQL-Consulting.com


# О чем сегодня будем говорить

- В каких местах системы могут быть проблемы
- Что можно сделать и какие гайки крутить
- Как искать узкие места


# Типичный веб проект

[client] $\Longleftrightarrow$ [web server] $\Longleftrightarrow$ [application] $\Longleftrightarrow$ [database]

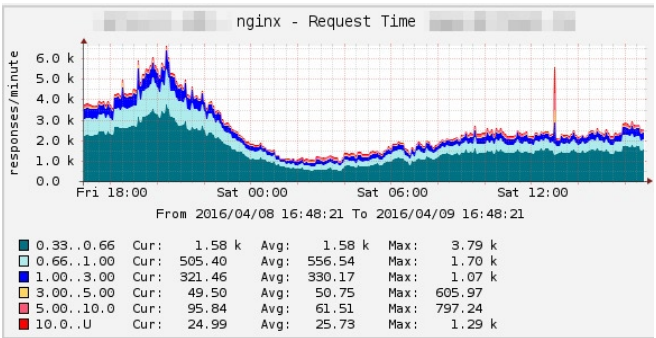

# Все ли у нас в порядке?

**Не все метрики одинаково полезны для оценки производительности**

- LA, CPU load, **%diskutil**, memory usage и т.п. нужны, но не всегда помогают
- Среднее время выполнения http запроса – как средняя температура по больнице
- Быстрый ответ конечно хорошо, но только если это не 5xx ошибка

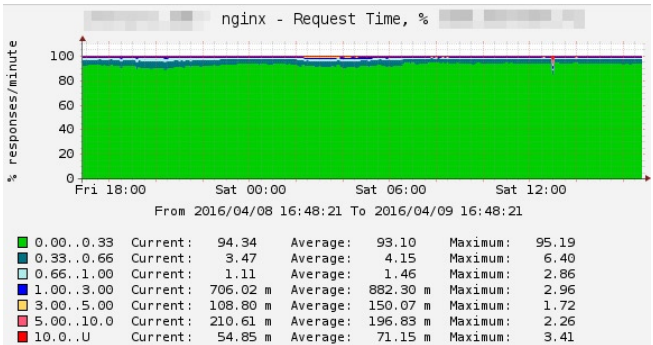
# Все ли у нас в порядке?

**График количества медленных (> 333ms) http запросов в минуту**



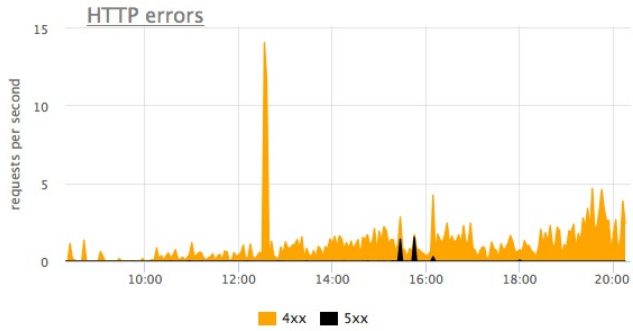
# Все ли у нас в порядке?

**График распределения http запросов по времени, в % в минуту**



# Все ли у нас в порядке?

**График количества 4xx и 5xx ошибок в секунду**




# Все ли у нас в порядке?

**Прежде чем крутить гайки стоит иметь какие-то метрики для оценки эффекта от изменений!**


# Где могут быть проблемы?

## Не модель OSI, но...

| | | | | |
|---|---|---|---|---|
| Application | “smart” ORM | | | long transactions |
| | unused data | queries inside loop | join 20+ tables | |
| Network | bandwidth | | | latency |
| Connection pooling | pgbouncer | | | |
| | mode | | | pool size |
| Database | config | | | |
| | memory | autovacuum | checkpointer | sync commit |
| Query | lack of indexes | | statistics | locks |
| System | kernel | io scheduler | huge pages | ionice/renice |
| Hardware | disks | RAID | RAM | NUMA |


# Длинные транзакции

- Очень плохо для базы из-за реализации multiversion concurrency control (MVCC)
- При каждом update строки создается ее копия
- Ненужные копии подчищаются процессом autovacuum
- Пока длинная транзакция открыта, автовакуум не может их почистить


# Длинные транзакции

- Приводят к распуханию (bloat) таблиц и индексов
- Запросы могут работать медленней из-за необходимости сканировать неактуальные версии строк
- Освободить уже занятое место не всегда просто

# Длинные транзакции

**Как бороться?**

- Мониторинг длины самой долгой транзакции (на репликах с hot_standby_feedback = on тоже!)
- Автоматически прибивать по крону (см. pg_terminate_backend(), pg_stat_activity)
- Разграничение пользователей по допустимому времени ответа
- Модифицировать приложение
- pg_dump ⇒ pg_basebackup


# ORM

- Для сложных выборок запросы лучше писать самостоятельно
- Не вызывать запросы в циклах без сильной необходимости
- Не стоит ожидать что запрос с 20 joins будет работать быстро
- Комментарии с ip/hostname/appname/stacktrace бывают полезны


# SQL запросы это не только select и join

- [recursive]CTE
- Window functions
- Lateral join
- DISTINCT ON
- EXISTS / NOT EXISTS
- generate_series()

- Arrays
- hstore/json/jsonb
- COPY
- Materialized views
- Unlogged tables
- pl/* functions

**Нет времени объяснять, надо использовать!**

# SQL запросы это не только select и join

## Выборка TOP N по списку

```
SELECT *
FROM (
    VALUES (29),(68),(45),(47),(50),(41),(11),(4),(83),(60)
) AS t(category_id),
LATERAL (
    SELECT * FROM posts WHERE posts.category_id = t.category_id ORDER BY created_at DESC LIMIT 5
) AS _t;
```


# SQL запросы это не только select и join

**Одним запросом к базе можно производить почти любые вычисления**


# Хватает ли сети?

**Latency**

- Оптимально, когда сервера подключены в один switch
- ping между приложением и базой $\approx$ 0.1ms
- Если приложение делает много запросов – то критичный параметр

# Хватает ли сети?

**Bandwidth**

- Расходы на репликацию
- pg_basebackup --max-rate
- Несколько интерфейсов, bonding, 10Gbps

# Connection pooling: pgbouncer


- 1 connect to DB = 1 process (postgresql backend)
- pool_mode = (session|transaction|statement)


# Connection pooling: pgbouncer

- pool_mode = transaction, если возможно
- Помним о сессионных переменных, prepared statements
- pool_size = (10|20|30)
- max_client_connections = (1000|10000)


# Какую версию PostgreSQL использовать?

- Поддерживаемые версии: 9.1-9.5
- Последняя минорная версия

# postgresql.conf

**shared_buffers**

- по-умолчанию 32MB/128MB
- 25% доступной RAM – хорошая отправная точка
- 75% – может быть хорошо, если база помещается в память

# postgresql.conf

## Двойное кэширование

# postgresql.conf

- work_mem – внутренняя память процесса для сортировки/hash таблицы. по-умолчанию 1MB/4MB
- maintenance_work_mem
- effective_cache_size – подсказка планировщику о размере кэша

# postgresql.conf

**autovacuum**

- autovacuum_vacuum_scale_factor по-умолчанию 0.2 (20% таблицы)
- autovacuum_analyze_scale_factor по-умолчанию 0.1 (10% таблицы)
- autovacuum_max_workers
- autovacuum_vacuum_cost_delay

# postgresql.conf

## WAL

- synchronous_commit = on (можно выключить, если не справляются диски, но нужно понимать последствия)
- wal_writer_delay = 200ms..10s
- fsync = on (не выключать!)

# postgresql.conf

**checkpointer**

- checkpoint_segments (до 9.5)
- min_wal_size/max_wal_size (9.5+)
- checkpoint_timeout
- checkpoint_completion_target


# Как искать проблемные запросы?

- логгирование запросов вместе с временем выполнения через log_min_duration_statement
- парсинг логов через pgfouine, pgbadger, loganalyzer
- pg_stat_statements (9.2+)


# Как искать проблемные запросы?

```
pgday=# select * from (select unnest(proargnames) from pg_proc where proname = 'pg_stat_statements')
       unnest
---------------------
 userid
 dbid
 query
 calls
 total_time
 rows
 ...
 blk_read_time
 blk_write_time
```


# Как искать проблемные запросы?

- track_io_timing = on (на экзотических платформах проверить overhead через pg_test_timing)
- track_functions = (none|pl|all)
- track_activity_query_size
- pg_stat_statements.max = 10000
- pg_stat_statements.track = (top|all)
- pg_stat_statements.track_utility = off
- pg_stat_statements_reset()


# Как искать проблемные запросы?

## sql/global_reports/query_stat_total.sql

```
total time: 82:08:45 (IO: 1.56%)
total queries: 3,366,257,532 (unique: 9,072)
report for all databases, version 0.9.3 @ PostgreSQL 9.5.2
tracking top 10000 queries, logging 100ms+ queries

================================================================================================
pos:1  total time: 20:42:35 (25.2%, CPU: 25.6%, IO: 0.0%)  calls: 1,824 (0.00%)
                                                           avg_time: 40874.96ms (IO: 0.0%)
user: bravo  db: echo  rows: 96,797,801,178  query:
SELECT * FROM oscar_recent WHERE id > ?
```


# Как ускорить запрос?

- Достаем из логов параметры запроса
- Выполняем explain analyze запроса с данными параметрами
- Смотрим на план, медитируем
- В сложных случаях смотрим на что тратится время на explain.depesz.com
- Если не хватает индексов – добавляем
- Если планировщик не прав, пробуем получить другие планы


# Как ускорить запрос?

```
                                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on oscar_recent  (cost=0.00..855857.35 rows=62938380 width=42)
                           (actual time=0.018..9436.857 rows=63020558 loops=1)
   Filter: (id > '3244145575'::bigint)
 Planning time: 0.093 ms
 Execution time: 11188.941 ms
```

# Как ускорить запрос?

**Методы получения данных**

- seq scan - последовательное чтение таблицы
- index scan - random io (чтение индекса + чтение таблицы)
- index only scan (9.2+)[1]
- bitmap index scan - компромисс между seq scan/index scan, возможность использования нескольких индексов в OR/AND условиях

[1] https://wiki.postgresql.org/wiki/Index-only_scans


# Как ускорить запрос?

**Методы соединения данных**

- nested loop - оптимален для небольших наборов данных
- hash join - оптимален для больших наборов данных
- merge join - оптимален для больших наборов данных, в случае, если они отсортированы

# Как ускорить запрос?

## Какие бывают индексы?

- partial

```
create index concurrently ... on post using btree(domain_id, created)
where pinned = true;
```

- multicolumn

```
create index concurrently ... on events using btree(user_id, type);
```

- functional

```
create index concurrently ... on i_movement
using btree((coalesce(m_movement_id, 0)));
```

- btree/gin/gist/brin

# Система

- Linux: Debian/Ubuntu/CentOS/RHEL
- в kernel 3.2 есть некоторые проблемы с IO[2]
- I/O scheduler: noop, deadline, cfq
- ionice/renice background процессам

[2] http://www.databasesoup.com/2014/09/why-you-need-to-avoid-linux-kernel-32.html

# Система

**sysctl.conf**

- по-умолчанию vm.dirty_ratio = 20, vm.dirty_background_ratio = 10
- vm.dirty_bytes
- vm.dirty_background_bytes
- vm.swappiness = 1 (swap лучше иметь, но он не должен использоваться)

# Файловая система

- ext4/xfs
- noatime
- barrier=0 (при наличии raid контроллера с "батарейкой")

Стачка
2016

## Диски

- SSD (server grade!)
- SAS 15k
- SATA


# RAID

- RAID 10
- контроллер с "батарейкой"(BBU)
- cache mode write back

# RAM

- В один сервер можно поставить сравнительно много 128GB-256GB-...
- Хорошо, когда активно используемая часть базы помещается в память
- Для больших объемов имеет смысл включить huge pages (9.2, 9.4+)[3]

[3] https://habrahabr.ru/post/228793/

# CPU

- Обычно не является лимитирующим фактором, но не всегда
- Для многопроцессорных систем следует выключать NUMA[4]:
- numa → off (node interleaving → enabled) в BIOS
- или vm.zone_reclaim_mode = 0 в sysctl.conf

[4] http://frosty-postgres.blogspot.ru/2012/08/postgresql-numa-and-zone-reclaim-mode.html


# Заключение

- Нужны метрики производительности системы
- Потенциальных узких мест в системе может быть много
- Возможности по обработке данных у SQL запросов очень большие
- Для поиска проблемных запросов парсим логи или используем pg_stat_statements
- Для оптимизации запросов нужно уметь читать вывод explain
- К выбору железа нужно подходить с умом


# Полезные ссылки

- Different Approaches for MVCC used in well known Databases
- depesz: Explaining the unexplainable
- Объясняя необъяснимое
- `https://github.com/PostgreSQL-Consulting/pg-utils`
- `http://blog.postgresql-consulting.com/`


# Вопросы?

alexey.ermakov@postgresql-consulting.com

| | | | | | |
|---|---|---|---|---|---|
| Application | “smart” ORM | | | long transactions | |
| | unused data | queries inside loop | join 20+ tables | | |
| Network | bandwidth | | | latency | |
| Connection pooling | pgbouncer | | | | |
| | mode | | | pool size | |
| Database | config | | | | |
| | memory | autovacuum | | checkpointer | sync commit |
| Query | lack of indexes | | | statistics | locks |
| System | kernel | io scheduler | | huge pages | ionice/renice |
| Hardware | disks | RAID | | RAM | NUMA |