Промышленный подход к тюнингу PostgreSQL: эксперименты над базами данных

Николай Самохвалов

https://postgres.ai

email: nik@postgres.ai

twitter: @postgresmen

О докладчике

- Опыт Postgres с 2005 (РСУБД — с 2001)
- Co-founder/CTO нескольких компаний (везде — Postgres)
- Co-founder #RuPostgres (1900+ человек на Meetup, 2-е место в мире)
- ПК различных конференций (BackendConf, Highload++, etc) – с 2007

О докладчике

- Опыт Postgres с 2005 (РСУБД — с 2001)
- Co-founder/CTO нескольких компаний (везде — Postgres)
- Co-founder #RuPostgres (1900+ человек на Meetup, 2-е место в мире)
- ПК различных конференций (BackendConf, Highload++, etc) – с 2007

PostgreSQL-консалтинг в SF Bay Area

О докладчике

- Опыт Postgres с 2005 (РСУБД — с 2001)
- Co-founder/CTO нескольких компаний (везде — Postgres)
- Co-founder #RuPostgres (1900+ человек на Meetup, 2-е место в мире)
- ПК различных конференций (BackendConf, Highload++, etc) – с 2007

PostgreSQL-консалтинг в SF Bay Area

↓

Postgres.ai – платформа для Postgres DBA,
автоматизация тех задач,
которые ещё не автоматизированы
«облаками»

О докладе

В докладе **не** будет:

О докладе

В докладе **не** будет:

- «серебряных пуль», волшебных настроек, универсальных рецептов;

О докладе

В докладе **не** будет:

- «серебряных пуль», волшебных настроек, универсальных рецептов;
- хардкорных «внутренностей».

О докладе

В докладе **не** будет:

- «серебряных пуль», волшебных настроек, универсальных рецептов;
- хардкорных «внутренностей».

А что будет?

О докладе

В докладе **не** будет:

- «серебряных пуль», волшебных настроек, универсальных рецептов;
- хардкорных «внутренностей».

А что будет?

- оптимизация с помощью экспериментов: принципы, идеи и инструменты, которые можно применять на практике;

О докладе

В докладе **не** будет:

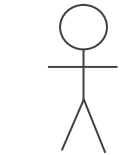
- «серебряных пуль», волшебных настроек, универсальных рецептов;
- хардкорных «внутренностей».

А что будет?

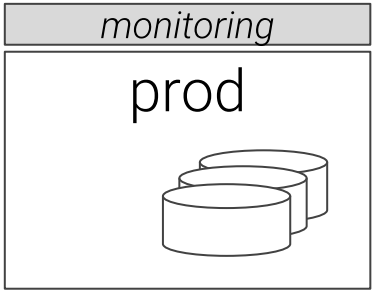
- оптимизация с помощью экспериментов: принципы, идеи и инструменты, которые можно применять на практике;
- опыт использования в разных компаниях, от маленьких стартапов до компаний-«единорогов»

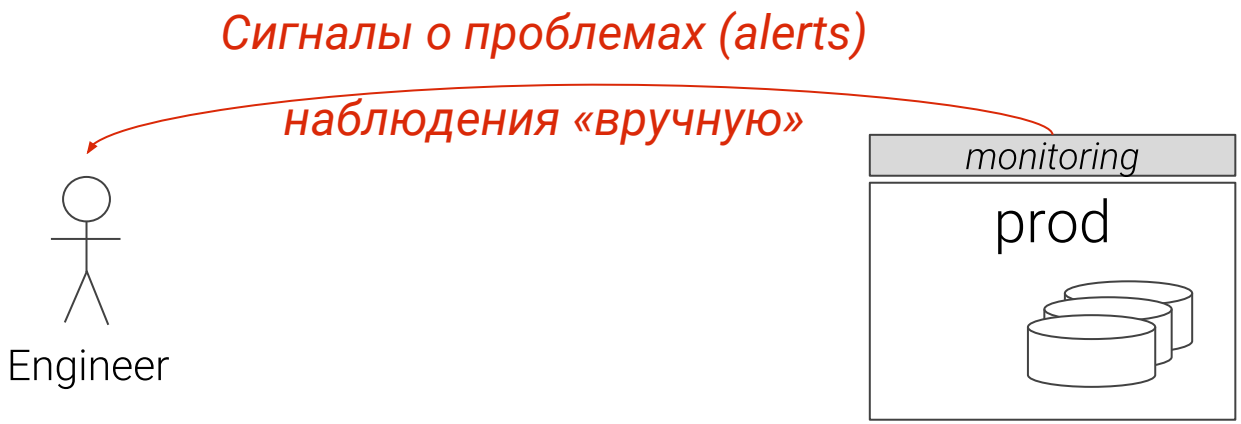
Postgres.ai:
от идей к компании и платформе

Как разработчики/DBA находят проблемы производительности сегодня



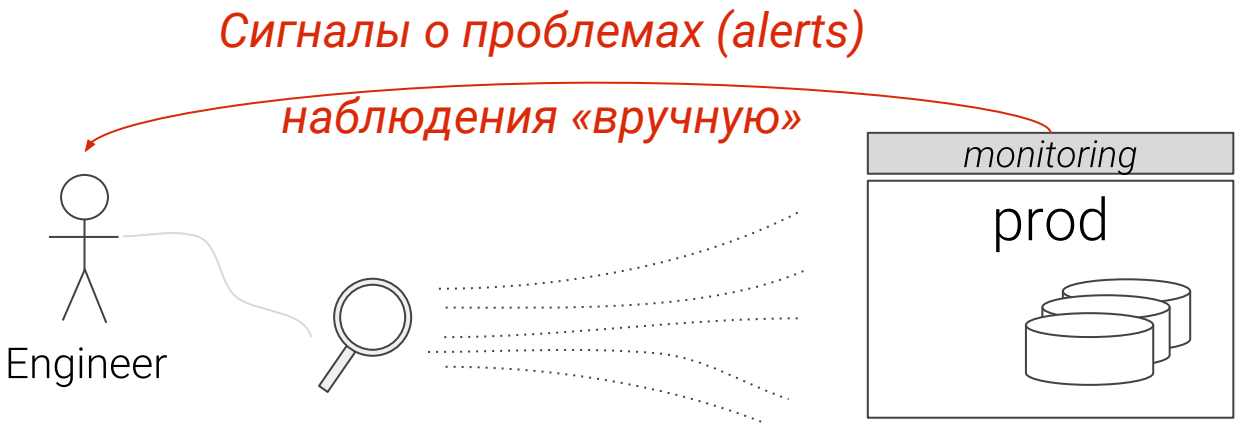
Engineer



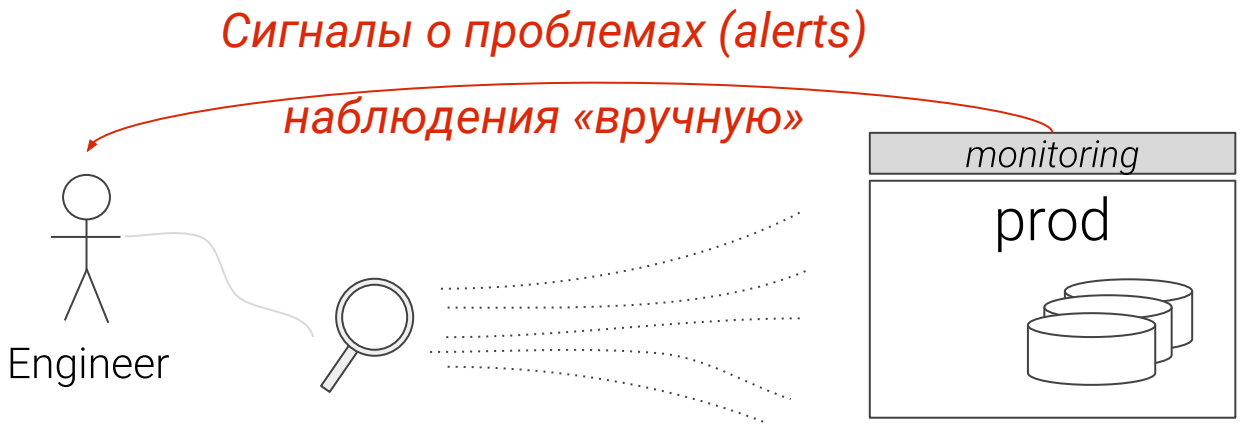

Сигналы о проблемах (alerts)
наблюдения «вручную»
Engineer
monitoring
prod

Сигналы о проблемах (alerts)

наблюдения «вручную»

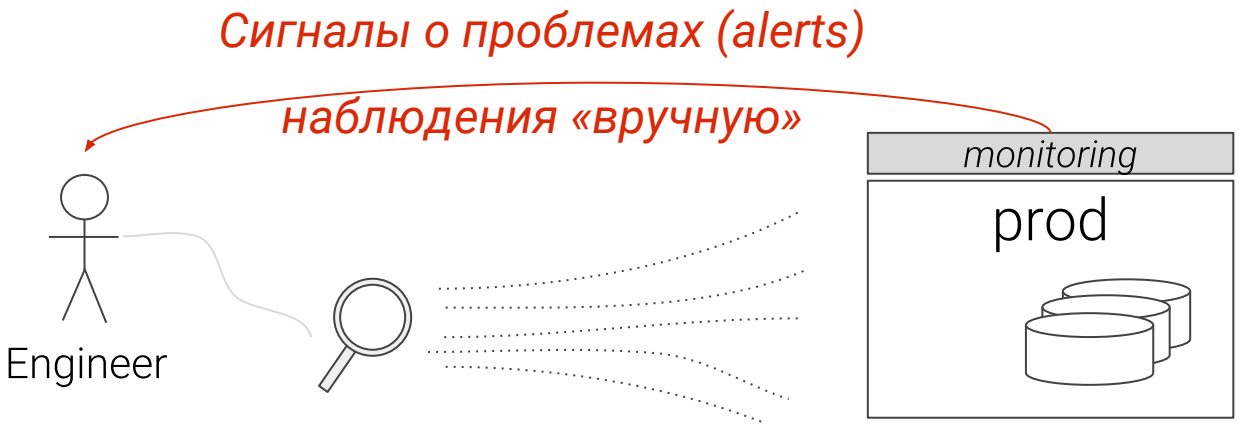


Наиболее популярные средства для анализа запросов «в целом»:



Наиболее популярные средства для анализа запросов «в целом»:

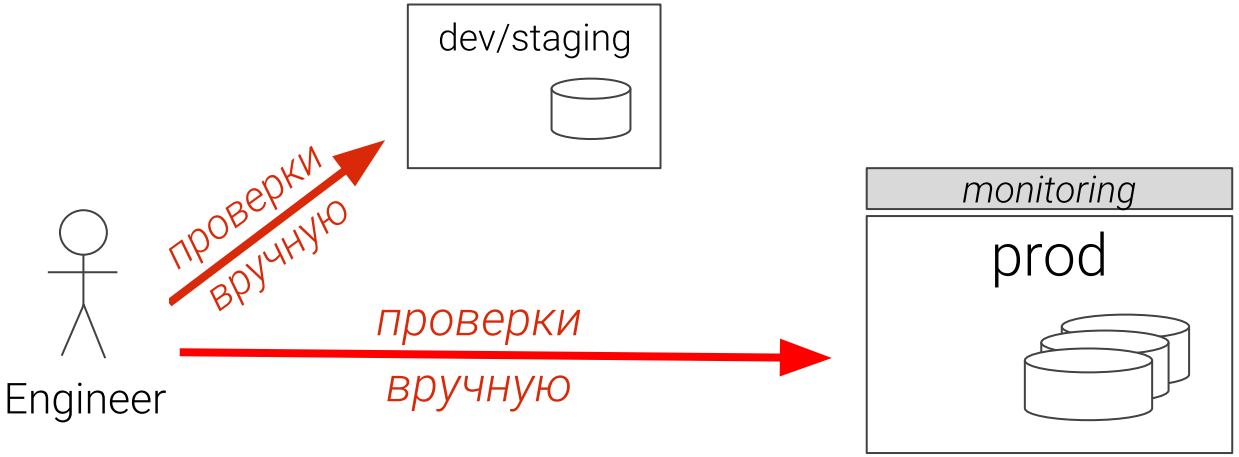
- pg_stat_statements
  - нет экземпляров запросов
  - нет планов



Наиболее популярные средства для анализа запросов «в целом»:

- pg_stat_statements
  - нет экземпляров запросов
  - нет планов
- log analysis (pgBadger)
  - требует поддержки
  - не «realtime»
  - обычно нет планов (есть, если auto_explain)
  - часто неполноценная картина (`log_min_duration_statement >> 0`)

Как DBA решают найденные проблемы?


dev/staging
проверки
вручную
monitoring
prod
Engineer
проверки
вручную

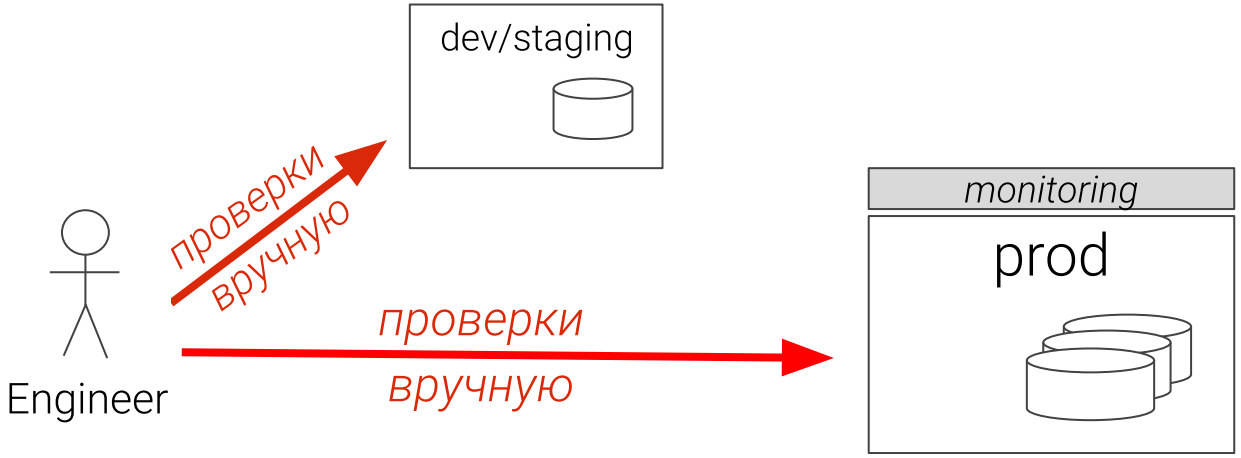
4 способа улучшить производительность:

4 способа улучшить производительность:

1. Тюнинг конфигурации Postgres

4 способа улучшить производительность:

1. Тюнинг конфигурации Postgres
2. Добавить/удалить индексы



4 способа улучшить производительность:

1. Тюнинг конфигурации Postgres
2. Добавить/удалить индексы
3. Изменить SQL-запрос / схему БД

4 способа улучшить производительность:

1. Тюнинг конфигурации Postgres
2. Добавить/удалить индексы
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

4 способа улучшить производительность:

1. Тюнинг конфига Postgres
2. Индексы
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

4 способа улучшить производительность:

1. Тюнинг конфига Postgres ~280 knobs!
2. Индексы
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

4 способа улучшить производительность:

1. Тюнинг конфига Postgres

~280 knobs!

2. Индексы

btree, hash, GiST, SP-GiST, GIN,
RUM, BRIN, Bloom;
unique, partial, functional, covering

3. Изменить SQL-запрос / схему БД

4. Нарастить ресурсы (CPU, RAM, disks)

4 способа улучшить производительность:

1. Тюнинг конфига Postgres
2. Индексы
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

~280 knobs!

btree, hash, GiST, SP-GiST, GIN, RUM, BRIN, Bloom;
unique, partial, functional, covering

No real-workload and real-data verification

(or very limited and affecting production)

4 способа улучшить производительность:

1. Тюнинг конфига Postgres
2. Индексы
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

~280 knobs!

btree, hash, GiST, SP-GiST, GIN, RUM, BRIN, Bloom;
unique, partial, functional, covering

No real-workload and real-data verification

(or very limited and affecting production)

Sub-optimal or even far from optimal decisions

4 способа улучшить производительность:

1. Тюнинг конфига Postgres — ~280 knobs!
2. Индексы — btree, hash, GiST, SP-GiST, GIN, RUM, BRIN, Bloom; unique, partial, functional, covering
3. Изменить SQL-запрос / схему БД
4. Нарастить ресурсы (CPU, RAM, disks)

No real-workload and real-data verification

(or very limited and affecting production)

Sub-optimal or even far from optimal decisions

DBA-эксперт пропускает многие шаги

«Чёрная магия»!

Примеры из жизни

Почитали умные статьи или посты в блогах. Возникла идея:

Значение `default_statistics_target` (100) слишком мало. Давайте поменяем на 1000!

...Отличная идея!

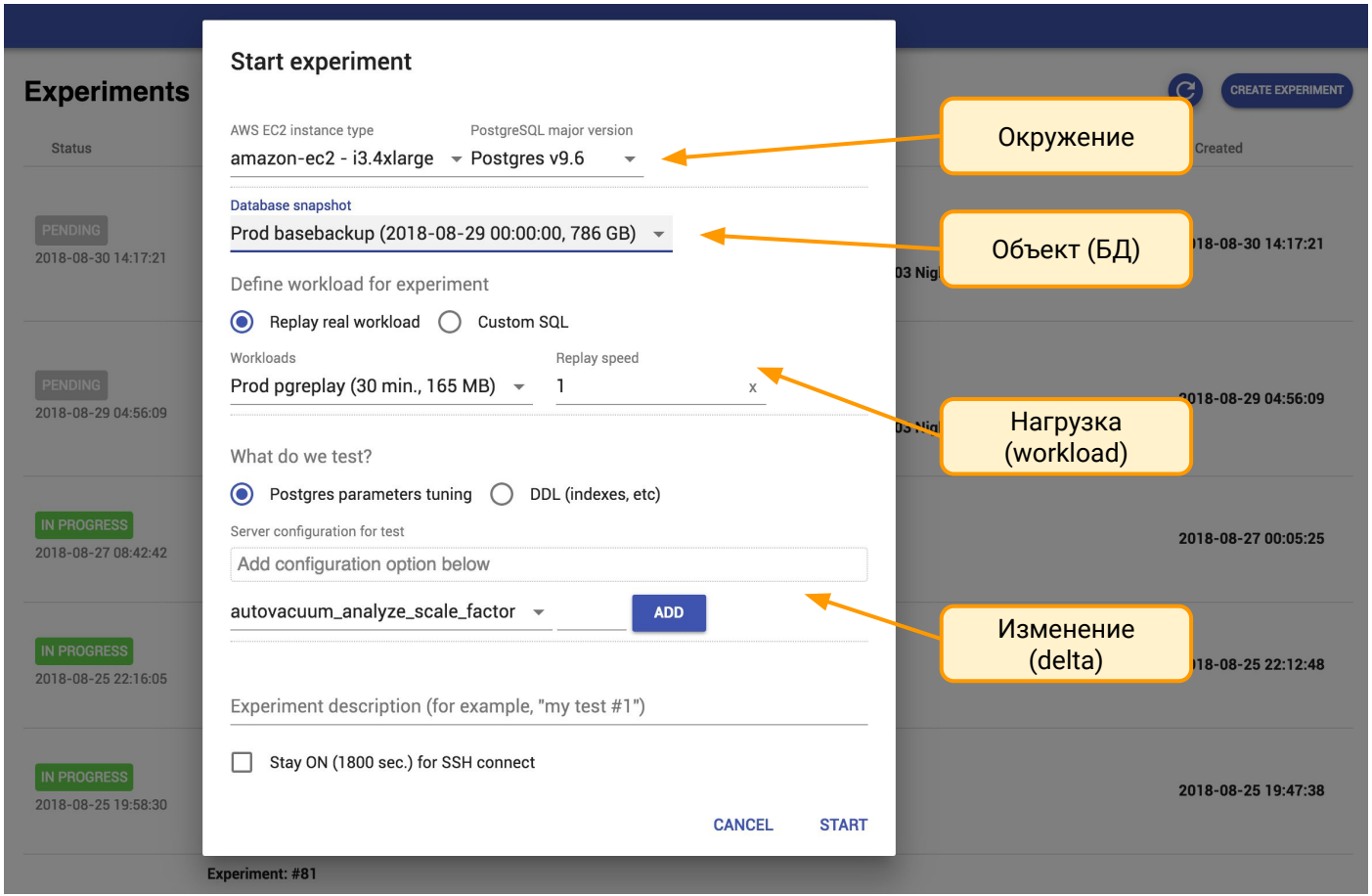
...Сделано — уже в production!

Но все ли запросы улучшились?
Как именно изменился каждый запрос?

Но все ли запросы улучшились?
Как именно изменился каждый запрос?

Проверим с помощью экспериментов!

Postgres.ai — GUI: форма создания эксперимента



2 года спустя: проверка с помощью эксперимента, значения 100 и 1000:

Experiment #69

On i3.xlarge with snapshot: '2018-07-12 Prod' of database 'postila_ru' from server 'dev.imgdata.ru'. Postgres version: 9.6

Experiment status: DONE Changed: 2018-07-22 07:29:59

Experiment created: 2018-07-20 21:52:50

Run ID: #69_94.

Status: done

Total query duration: 45m46s651ms249µs

Queries: 2208

Query groups : 98

Errors: 12

Number of errors: 12

Number of unique normalized events: 2

Show more

Run ID: #69_95.

Status: done

Delta: `default_statistics_target = 1000`

Total query duration: 42m5s361ms43µs

-3m41s290ms206µs / -8.06% / 1.09 times

Queries: 2208

Query groups : 97

Errors: 12

Number of errors: 12

Number of unique normalized events: 2

Show more

Queries

Query group #1

```
SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t0_c1", "t"."parentId" AS "t0_c2", "t"."userId" AS "t0_c3", "t"."boardId" AS "t0_c4", "t"."s
ource" AS "t0_c5", "t"."sourceDomain" AS "t0_c6", "t"."channel" AS "t0_c7", "t"."message" AS "t0_c8", "t"."likes" AS "t0_c9", "t"."reposts"
AS "t0_c10", "t"."comments" AS "t0_c11", "t"."created" AS "t0_c12", "t"."newsletter" AS "t0_c13", "t"."nl_complete" AS "t0_c14", "t"."nl_las
t_id" AS "t0_c15", "t"."changed" AS "t0_c16", "t"."tsvector" AS "t0_c17", "t"."creation_method" AS "t0_c18", "t"."pushed_to_facebook" AS "t0
```

Experiment #69

ДО: `default_statistics_target = 100`

ПОСЛЕ: `default_statistics_target = 1000`

On … m serve

Exp…

Experiment created: 2018-07-20 21:52:50

Run ID: #69_94.

Status: done

Total query duration: 45m46s651ms249µs

Queries: 2208

Query groups : 98

Errors: 12

Number of errors: 12

Number of unique normalized events: 2

Show more

Run ID: #69_95.

Status: done

Delta: `default_statistics_target = 1000`

Total query duration: 42m5s361ms43µs

-3m41s290ms206µs / -8.06% / 1.09 times

Queries: 2208

Query groups : 97

Errors: 12

Number of errors: 12

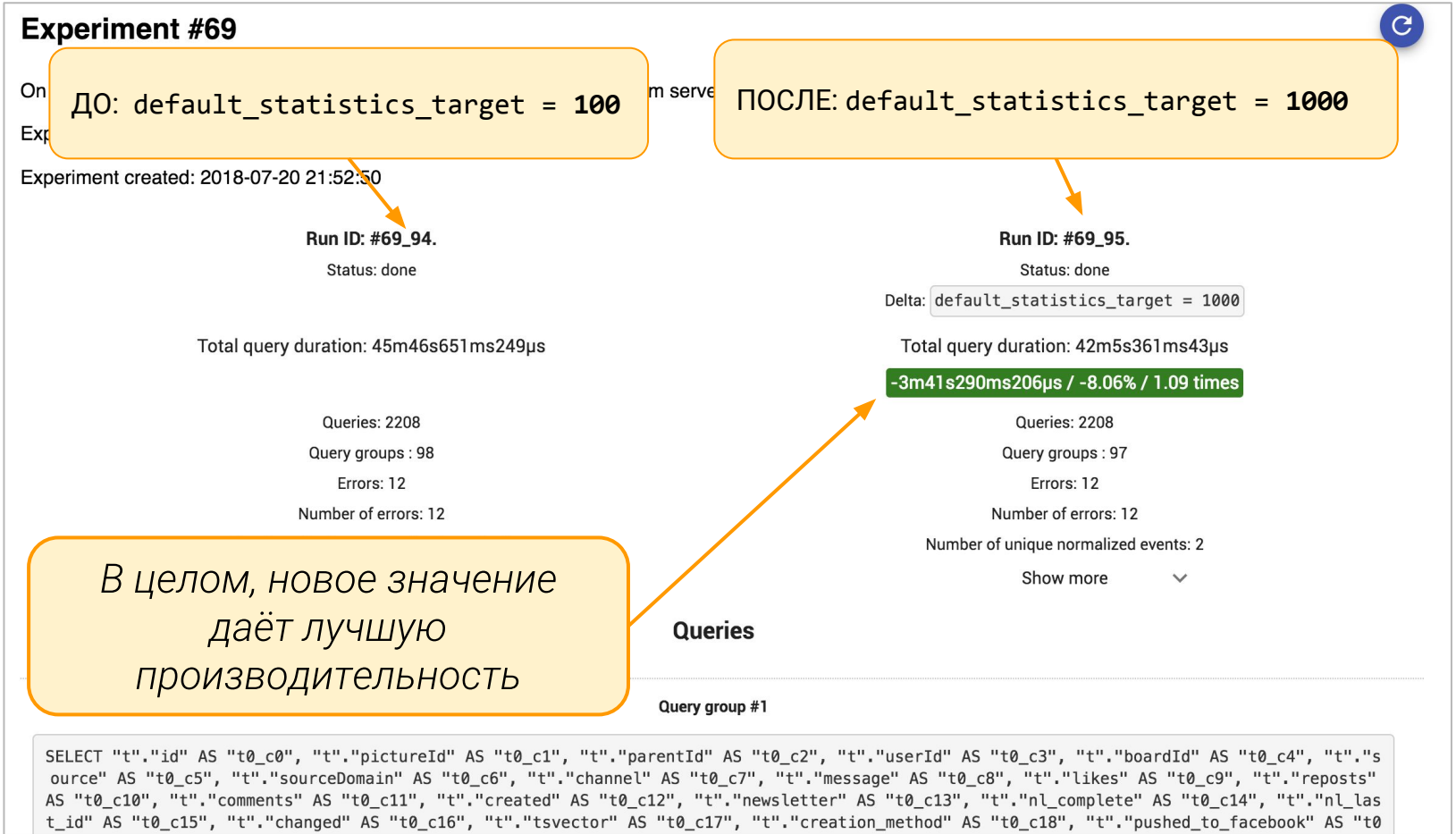
Number of unique normalized events: 2

Show more

Queries

Query group #1

```
SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t0_c1", "t"."parentId" AS "t0_c2", "t"."userId" AS "t0_c3", "t"."boardId" AS "t0_c4", "t"."s
ource" AS "t0_c5", "t"."sourceDomain" AS "t0_c6", "t"."channel" AS "t0_c7", "t"."message" AS "t0_c8", "t"."likes" AS "t0_c9", "t"."reposts"
AS "t0_c10", "t"."comments" AS "t0_c11", "t"."created" AS "t0_c12", "t"."newsletter" AS "t0_c13", "t"."nl_complete" AS "t0_c14", "t"."nl_las
t_id" AS "t0_c15", "t"."changed" AS "t0_c16", "t"."tsvector" AS "t0_c17", "t"."creation_method" AS "t0_c18", "t"."pushed_to_facebook" AS "t0
```


Experiment #69
ДО: default_statistics_target = 100
ПОСЛЕ: default_statistics_target = 1000
Experiment created: 2018-07-20 21:52:50
Run ID: #69_94.
Status: done
Total query duration: 45m46s651ms249µs
Queries: 2208
Query groups : 98
Errors: 12
Number of errors: 12
Run ID: #69_95.
Status: done
Delta: default_statistics_target = 1000
Total query duration: 42m5s361ms43µs
-3m41s290ms206µs / -8.06% / 1.09 times
Queries: 2208
Query groups : 97
Errors: 12
Number of errors: 12
Number of unique normalized events: 2
Show more
В целом, новое значение даёт лучшую производительность
Queries
Query group #1
SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t0_c1", "t"."parentId" AS "t0_c2", "t"."userId" AS "t0_c3", "t"."boardId" AS "t0_c4", "t"."s
ource" AS "t0_c5", "t"."sourceDomain" AS "t0_c6", "t"."channel" AS "t0_c7", "t"."message" AS "t0_c8", "t"."likes" AS "t0_c9", "t"."reposts"
AS "t0_c10", "t"."comments" AS "t0_c11", "t"."created" AS "t0_c12", "t"."newsletter" AS "t0_c13", "t"."nl_complete" AS "t0_c14", "t"."nl_las
t_id" AS "t0_c15", "t"."changed" AS "t0_c16", "t"."tsvector" AS "t0_c17", "t"."creation_method" AS "t0_c18", "t"."pushed_to_facebook" AS "t0

ДО: `default_statistics_target = 100`

ПОСЛЕ: `default_statistics_target = 1000`

Query g

```
rentId"
main" A
```

Total duration: 3m28s597ms709µs

Count: 10

Total duration: 2m57s987ms259µs

-30s610ms450µs / -14.67 % / 1.17 times

Count: 10

Промотаем немного вниз...

Query group #4

```
SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t0_c1", "t"."parentId" AS "t0_c2", "t"."userId" AS "t0_c3", "t"."boardId" AS "t0_c4", "t"."s
                                        ource" AS "t0_c5", "t"."sourceDomain" AS "t0_c6",...
```

Total duration: 2m2s205ms534µs

Count: 86

Mean time: 1s420ms995µs

Max duration: 7s749ms642µs

Min duration: 697µs

Samples

Total duration: 3m50s716ms819µs

+1m48s511ms285µs / +88.79 %

Count: 86

Mean time: 2s682ms754µs

Max duration: 2m27s462ms230µs

Min duration: 1ms81µs

Samples

ДО: `default_statistics_target = 100`

ПОСЛЕ: `default_statistics_target = 1000`

Эта группа запросов стала **намного** медленнее после изменения!

Решили с помощью:
“`ALTER TABLE/INDEX ... ALTER COLUMN SET STATISTICS ...`“
на конкретном столбце

Query

rentId"
main" A

Total duration: 3m28s597ms709µs

Total duration: 2m57s987ms259µs

-30s610ms450µs / -14.67 % / 1.17 times

Count: 10

Mean time: 17s798ms726µs

Max duration: 1m44s620ms546µs

Min duration: 184ms883µs

Samples

ry group #4

```
SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t0_c1", "t"."parentId" AS "t0_c2", "t"."userId" AS "t0_c3", "t"."boardId" AS "t0_c4", "t"."s
                                        ource" AS "t0_c5", "t"."sourceDomain" AS "t0_c6",...
```

Total duration: 2m2s205ms534µs

Count: 86

Mean time: 1s420ms995µs

Max duration: 7s749ms642µs

Min duration: 697µs

Samples

Total duration: 3m50s716ms819µs

+1m48s511ms285µs / +88.79 %

Count: 86

Mean time: 2s682ms754µs

Max duration: 2m27s462ms230µs

Min duration: 1ms81µs

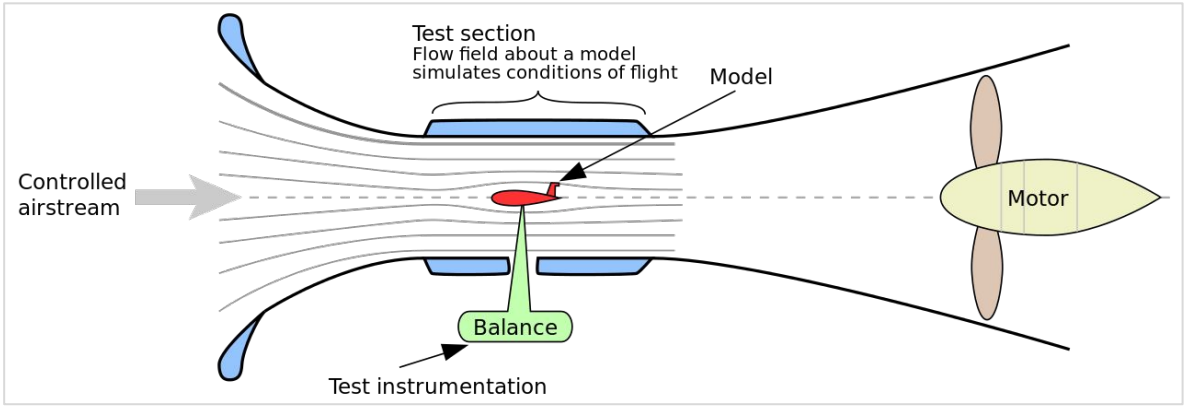
Samples

Управление изменениями в других областях

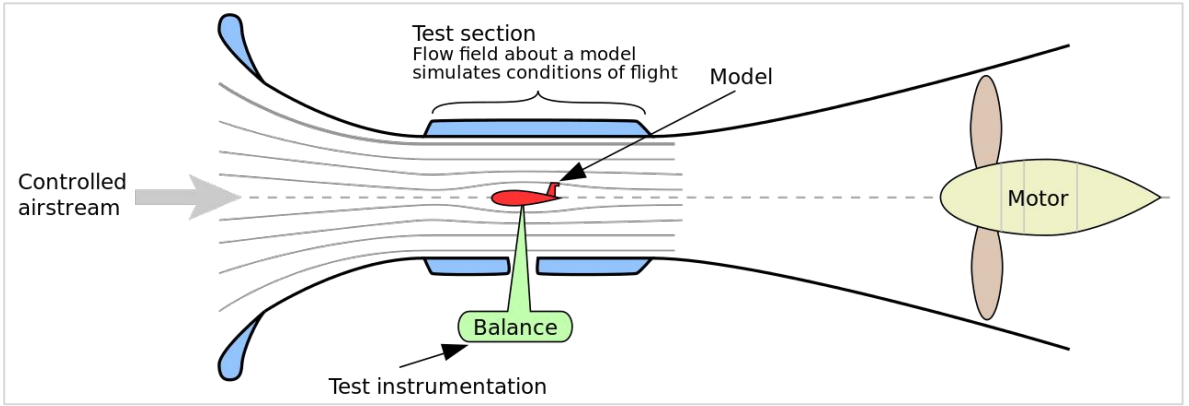
Интерфейсы (GUI, CLI, API — любые):

Большое количество развитых решений CI/CD

Ещё один пример



Ещё один пример



GFDL and CC-BY-2.5, Wikipedia.org

Авиация, космос, автомобилестроение и т.д.:

аэродинамическая труба

Эксперименты в развитых отраслях:

1. …производятся в специальном окружении,
   не в production!
   (staging, “лаборатория”)
2. **Детальный анализ** за счёт спецсредств
3. Высокий уровень ***автоматизации***.
   Много “прогонов”. Серии. Дешёвый, быстрый запуск

Nancy CLI — фундамент «лаборатории БД»

Nancy CLI сегодня

- Open source: https://gitlab.com/postgres-ai-team/nancy

  Список параметров: https://gitlab.com/postgres-ai-team/nancy/blob/master/help/nancy_run.md

Nancy CLI сегодня

- Open source: https://gitlab.com/postgres-ai-team/nancy

  Список параметров: https://gitlab.com/postgres-ai-team/nancy/blob/master/help/nancy_run.md

- Где можно запускать:
  - `--run-on localhost`: любая машина с MacOS, Debian/Ubuntu или RHEL/Centos)
  - `--run-on aws`: удалённый запуск на временных AWS EC2 (споты, рекомендуется i3)

Nancy CLI сегодня

- Open source: https://gitlab.com/postgres-ai-team/nancy

  Список параметров: https://gitlab.com/postgres-ai-team/nancy/blob/master/help/nancy_run.md

- Где можно запускать:
  - `--run-on localhost`: любая машина с MacOS, Debian/Ubuntu или RHEL/Centos)
  - `--run-on aws`: удалённый запуск на временных AWS EC2 (споты, рекомендуется i3)
- Поддерживаемые версии Postgres:
  - 9.6
  - 10
  - 11

Nancy CLI сегодня

- «Объект»:
  - `--db-dump`: дамп / sql-файл
  - `--db-local-pgdata`: копия PGDATA
  - `--db-pgbench`: БД, сгенерированная с помощью pgbench

Nancy CLI сегодня

- «Объект»:
  - `--db-dump`: дамп / sql-файл
  - `--db-local-pgdata`: копия PGDATA
  - `--db-pgbench`: БД, сгенерированная с помощью pgbench
- «Нагрузка» (workload):
  - --workload-custom-sql: в произвольный SQL, в один поток;
  - --workload-real: эмуляция «реальной» нагрузки с помощью pgreplay (требуются логи с «прода»), многопоточно. Можно задавать «скорость» (--workload-real-replay-speed)
  - --workload-pgbench: эмуляция нагрузки, “crafted workload”

Nancy CLI сегодня

- Что проверяем:
  - `--delta-sql-do` / `--delta-sql-undo`: DDL (например, новый индекс), изменение настроек таблиц, произвольные БД-миграции, пересбор статистики (ANALYZE), борьба с раздутием (bloat) и т.д.
  - --delta-config: изменение одной или нескольких опций конфига PostgreSQL

Nancy CLI сегодня

- Что проверяем:
  - `--delta-sql-do` / `--delta-sql-undo`: DDL (например, новый индекс), изменение настроек таблиц, произвольные БД-миграции, пересбор статистики (ANALYZE), борьба с раздутием (bloat) и т.д.
  - --delta-config: изменение одной или нескольких опций конфига PostgreSQL
- Что в результате (набор артефактов):
  - Снапшоты `pg_stat_***` (`pg_stat_statements`, `pg_stat_kcache`, `pg_stat_bgwriter` и т.д.)
  - Логи Postgres (отдельно: подготовка, проигрывание нагрузки)
  - FlameGraphs / perf cpu
  - «Родная» выдача pgreplay или pgbench
  - Информация о системе
  - [WIP/MR exists] базовые проверки CPU и IO (sysbench)

Технические сложности*

---

* на опыте использования в ряде компаний,
размеры БД — от малого (десятки ГБ) до среднего (несколько ТБ)
тип нагрузки — OLTP (до 15k TPS)

Как создавать качественную нагрузку?

```
log_min_duration_statement = 0
```

Страхи `log_min_duration_statement = 0`

Как оценить поток лога при `log_min_duration_statement = 0` (быстро и ничего не меняя!):
https://gist.github.com/NikolayS/08d9b7b4845371d03e195a8d8df43408 (внимание на комментарии)

```
postgres=# with const(stats_since, "pg_stat_statements.max") as (
  select
    (select stats_reset from pg_stat_database where datname = current_database()),
    (select setting from pg_settings where name = 'pg_stat_statements.max')
)
select
  (select stats_since from const),
  (select now() - stats_since from const) stats_age,
  count(*) as query_groups,
  (select "pg_stat_statements.max" from const),
  sum(calls * length(query)) as total_bytes,
  sum(calls * length(query)) / extract(epoch from now() - (select stats_since from const)) as bytes_per_sec,
  sum(calls) / extract(epoch from now() - (select stats_since from const)) as entries_per_sec
from pg_stat_statements
;
-[ RECORD 1 ]----------+------------------------------
stats_since            | 2018-10-14 21:05:01.409838+00
stats_age              | 1 day 20:22:22.575579
query_groups           | 486
pg_stat_statements.max | 500
total_bytes            | 45733261483
bytes_per_sec          | 286293.502638455
entries_per_sec        | 802.853456789136
```

Всего ожидаем ~300 kB/s,
~800 записей в лог в секунду

Ho...

```
log_destination = syslog
logging_collector = off
```

или

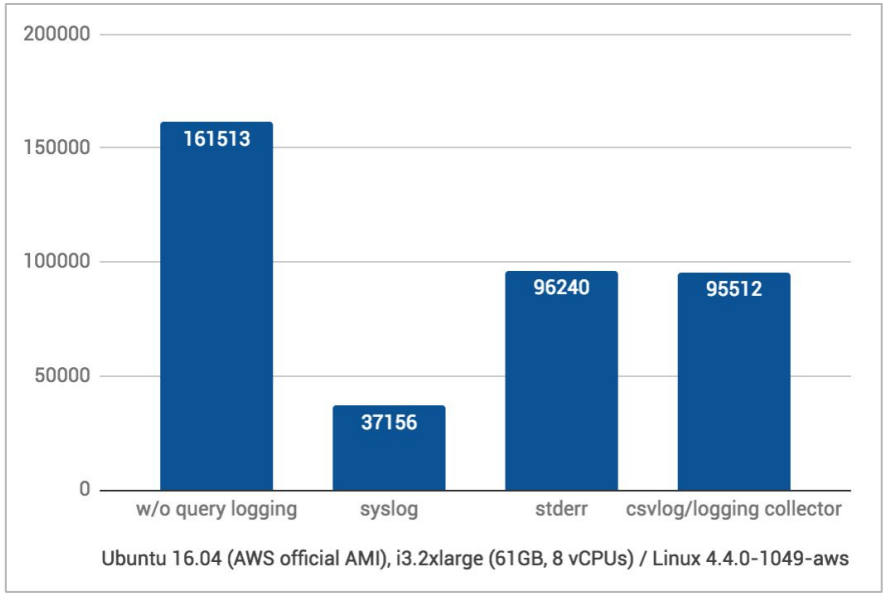
```
log_destination = stderr
logging_collector = off
```

или

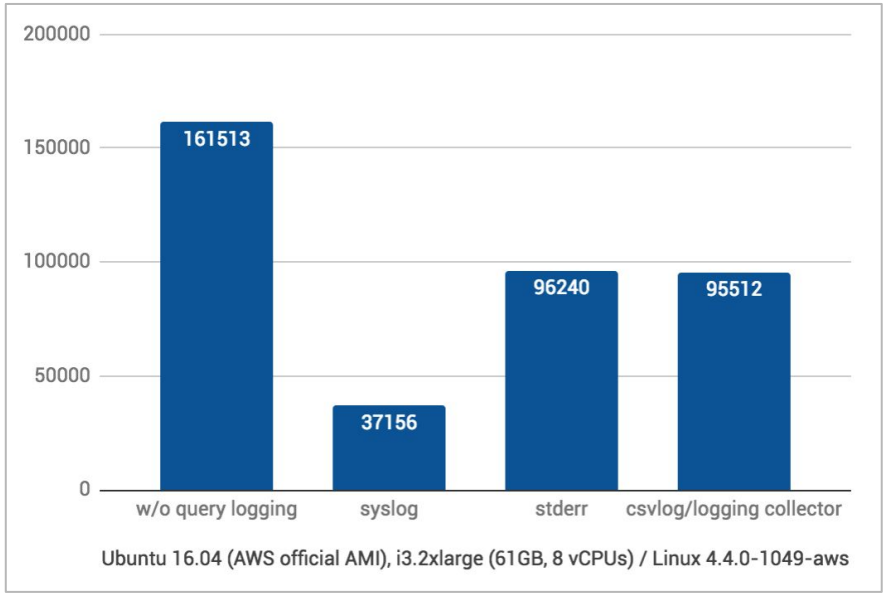
```
log_destination = csvlog
logging_collector = on
```

Какой вариант выбрать при интенсивном логировании?

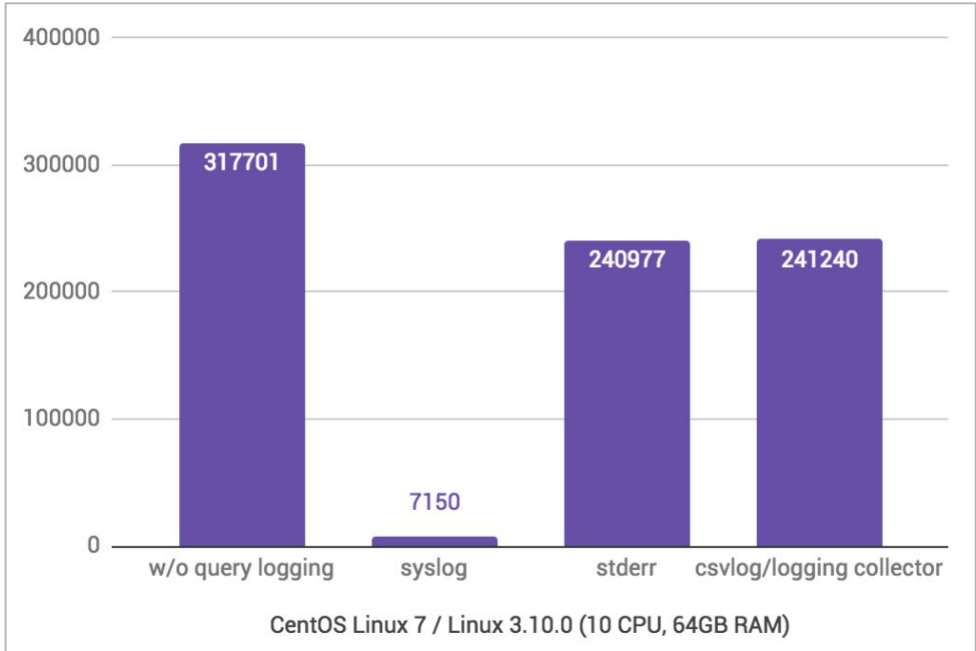
```
# Postgres 9.6.10
pgbench -U postgres -j2 -c24 -T60 -rnf - <<EOF
select;
EOF
```

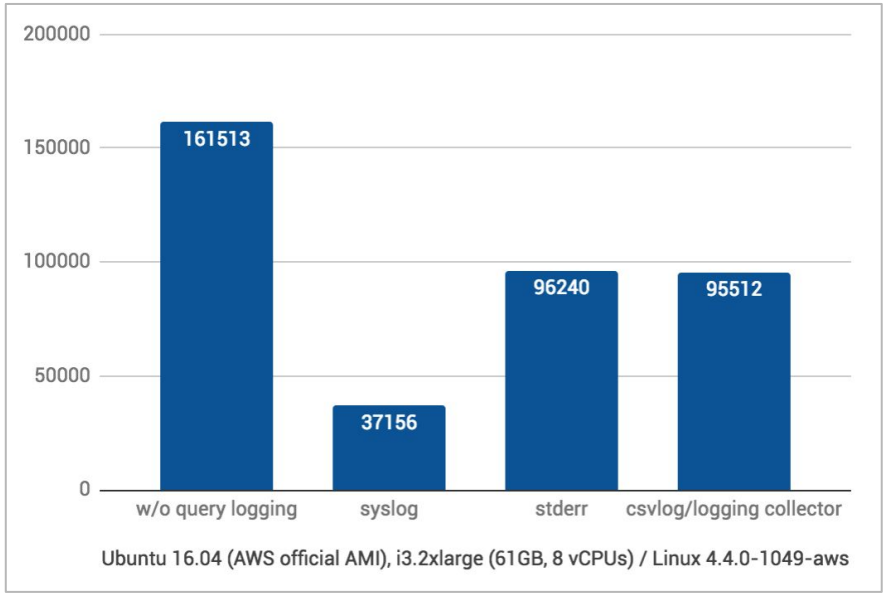



```
# Postgres 9.6.10
pgbench -U postgres -j2 -c24 -T60 -rnf - <<EOF
select;
EOF
```




```
# Postgres 9.6.10
pgbench -U postgres -j2 -c24 -T60 -rnf - <<EOF
select;
EOF
```



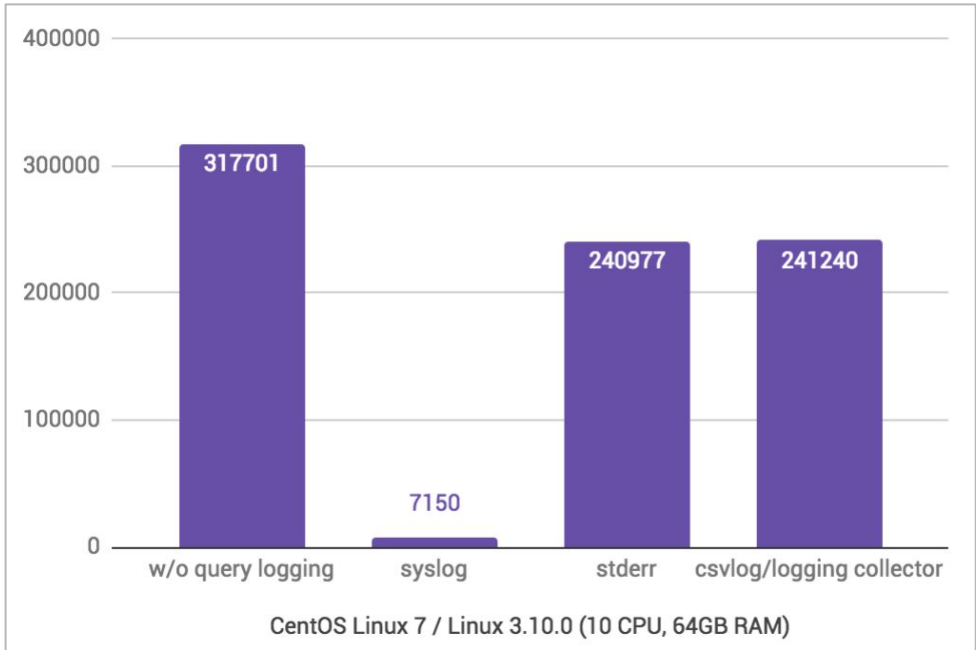



```
# Postgres 9.6.10
pgbench -U postgres -j2 -c24 -T60 -rnf - <<EOF
select;
EOF
```

All-queries logging with syslog is

- 44x slower compared to “no logging”
- 33x slower compared to stderr / logging collector

Be careful with syslog / journald



Советы для случая `log_min_duration_statement = 0`

- Оцените IOPS и поток записи
- Проверьте свою систему логирования (syslog/journald может замедлять всё радикально, подумайте о переходе на STDERR, logging collector)
- Если прогнозируемая нагрузка чрезмерно велика (десятки мегабайт и более), рассмотрите вариант с *сэмплированием* (`"SET log_min_duration_statement = 0;"` в конкретных, случайно выбранных сессиях)

Альтернативный вариант – “crafted workload”

`--workload-pgbench` (many thanks Michel Pelletier! @michelp)

Пример:

```
--workload-pgbench \
  "-n -c${CPU_COUNT} -j${CPU_COUNT} -T$T \
  -f /var/lib/postgresql/9.6/main/workload/1_select_posts.sql@6 \
  -f /var/lib/postgresql/9.6/main/workload/2_insert_post_view.sql@60 \
  -f /var/lib/postgresql/9.6/main/workload/3_insert_bot_visit.sql@50 \
  -f /var/lib/postgresql/9.6/main/workload/4_select_post_by_host.sql@2 \
  -f /var/lib/postgresql/9.6/main/workload/5_select_post_by_rare_host.sql@1 \
  -f /var/lib/postgresql/9.6/main/workload/6_select_post_by_category.sql@1 \
  -f /var/lib/postgresql/9.6/main/workload/7_conveyor.sql@6 \
  -f /var/lib/postgresql/9.6/main/workload/8_select_post_fresh.sql@1
...
```

Поможет «скрафтить нагрузку»:

postgres-checkup – периодическая диагностика здоровья БД

Open source: https://gitlab.com/postgres-ai-team/postgres-checkup

- Комплексный глубокий анализ. Мониторинги этого не делают
- Без вмешательства (лёгкие проверки)
- На наблюдаемые серверы ничего ставить не нужно (разве что pg_stat_statements, если ещё нет)

Любой «боевой» БД регулярные «чекапы» нужны не меньше, чем бэкапы

Собираем самые “влиятельные” группы запросов — отчёт K003 в postgres-checkup:

K003 Top 50 queries

Observations

Master (db2)

Start: 2019-01-25T21:15:40.689378+00:00
End: 2019-01-25T21:16:36.752667+00:00
Period seconds: 56.06329
Period age: 00:00:56.063289

Error (calls): 0.00 (0.00%) Error (total time): 0.00 (0.00%)

#	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	...	Query
1	27.00 0.48/sec 1.00/call 0.00%	2.729s 48ms/sec 101ms/call 14.00%	216.00 3.85/sec 8.00/call 2.00%	3.25M blks 57.81K blks/sec 120.04K blks/call 31.00%	0.00 blks 0.00 blks/sec 0.00 blks/call 0.00%	...	SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t
2	302.00 5.39/sec 1.00/call 5.00%	2.156s 38ms/sec 7ms/call 11.00%	302.00 5.39/sec 1.00/call 2.00%	2.60M blks 46.24K blks/sec 8.59K blks/call 25.00%	1.00 blks 0.02 blks/sec 0.00 blks/call 0.00%	...	SELECT s.lid, t.translation, s.version FROM loca

Собираем самые “влиятельные” группы запросов — отчёт K003 в postgres-checkup:

K003 Top 50 queries

Observations

Два снапшота pg_stat_statements, анализируем разницу основных метрик

Master (db2)

Start: 2019-01-25T21:15:40.689378+00:00
End: 2019-01-25T21:16:36.752667+00:00
Period seconds: 56.06329
Period age: 00:00:56.063289

Сортировка по total_time

Error (calls): 0.00 (0.00%) Error (total time): 0.00 (0.00%)

#	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	...	Query
1	27.00 0.48/sec 1.00/call 0.00%	2.729s 48ms/sec 101ms/call 14.00%	216.00 3.85/sec 8.00/call 2.00%	3.25M blks 57.81K blks/sec 120.04K blks/call 31.00%	0.00 blks 0.00 blks/sec 0.00 blks/call 0.00%	...	SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t
2	302.00 5.39/sec 1.00/call 5.00%	2.156s 38ms/sec 7ms/call 11.00%	302.00 5.39/sec 1.00/call 2.00%	2.60M blks 46.24K blks/sec 8.59K blks/call 25.00%	1.00 blks 0.02 blks/sec 0.00 blks/call 0.00%	...	SELECT s.lid, t.translation, s.version FROM loca

Собираем самые “влиятельные” группы запросов — отчёт K003 в postgres-checkup:

K003 Top 50 queries

Observations

Дифференцируем!
Два раза: по времени (период наблюдения) и по количеству вызовов в группе

Master (db2)

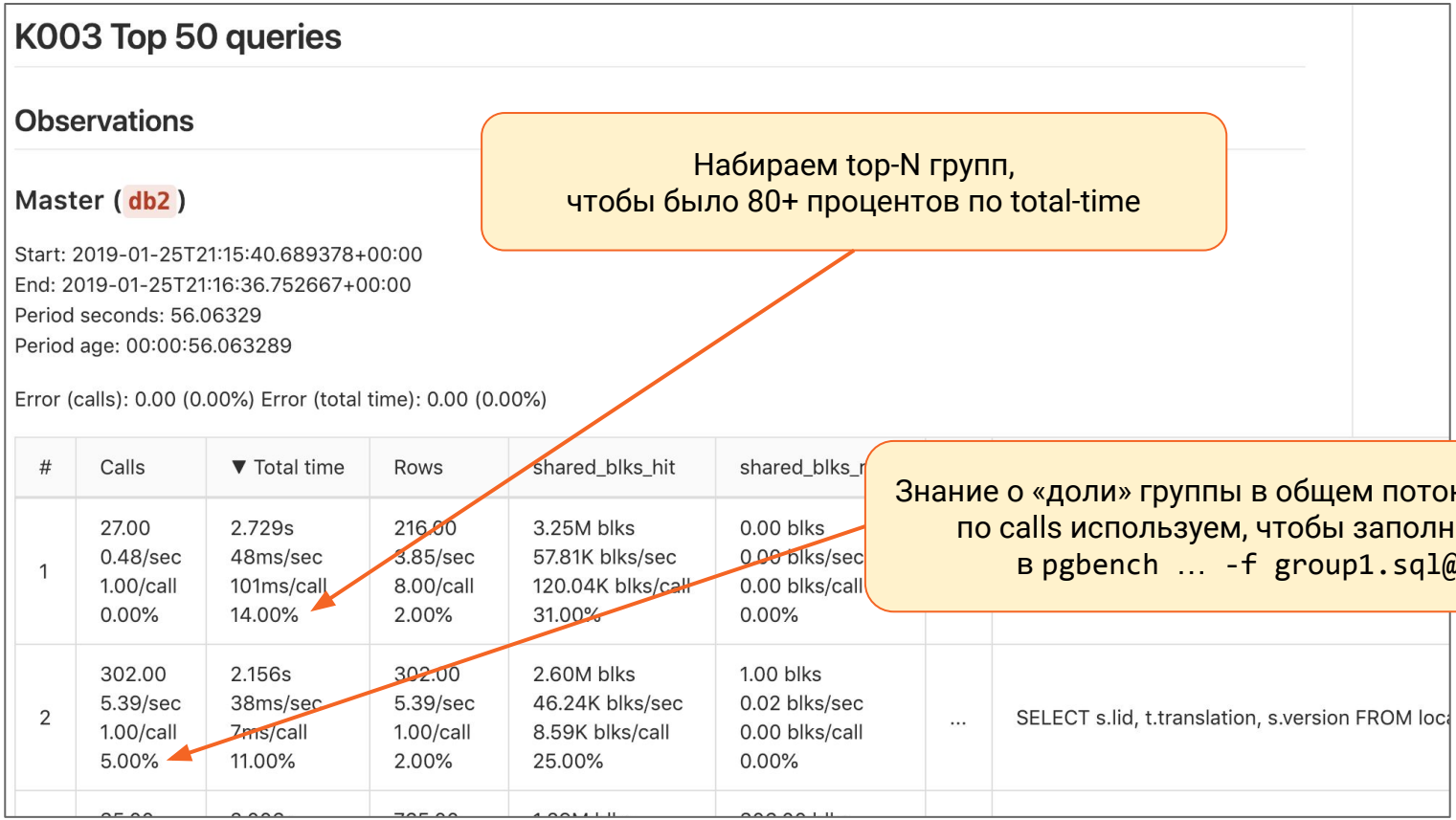
Start: 2019-01-25T21:15:40.689378+00:00
End: 2019-01-25T21:16:36.752667+00:00
Period seconds: 56.06329
Period age: 00:00:56.063289

Error (calls): 0.00 (0.00%) Error (total time): 0.00 (0.00%)

#	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	...	Query
1	27.00 0.48/sec 1.00/call 0.00%	2.729s 48ms/sec 101ms/call 14.00%	216.00 3.85/sec 8.00/call 2.00%	3.25M blks 57.81K blks/sec 120.04K blks/call 31.00%	0.00 blks 0.00 blks/sec 0.00 blks/call 0.00%	...	SELECT "t"."id" AS "t0_c0", "t"."pictureId" AS "t
2	302.00 5.39/sec 1.00/call 5.00%	2.156s 38ms/sec 7ms/call 11.00%	302.00 5.39/sec 1.00/call 2.00%	2.60M blks 46.24K blks/sec 8.59K blks/call 25.00%	1.00 blks 0.02 blks/sec 0.00 blks/call 0.00%	...	SELECT s.lid, t.translation, s.version FROM loca

А также смотрим на «удельный вес» группы, по каждой из метрик

Как именно «крафтить»



K003 Top 50 queries

Observations

Master (db2)

Start: 2019-01-25T21:15:40.689378+00:00
End: 2019-01-25T21:16:36.752667+00:00
Period seconds: 56.06329
Period age: 00:00:56.063289

Error (calls): 0.00 (0.00%) Error (total time): 0.00 (0.00%)

#	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_r		
1	27.00 0.48/sec 1.00/call 0.00%	2.729s 48ms/sec 101ms/call 14.00%	216.00 3.85/sec 8.00/call 2.00%	3.25M blks 57.81K blks/sec 120.04K blks/call 31.00%	0.00 blks 0.00 blks/sec 0.00 blks/call 0.00%		
2	302.00 5.39/sec 1.00/call 5.00%	2.156s 38ms/sec 7ms/call 11.00%	302.00 5.39/sec 1.00/call 2.00%	2.60M blks 46.24K blks/sec 8.59K blks/call 25.00%	1.00 blks 0.02 blks/sec 0.00 blks/call 0.00%	...	SELECT s.lid, t.translation, s.version FROM loc

Набираем top-N групп,
чтобы было 80+ процентов по total-time

Знание о «доли» группы в общем потоке запросов,
по calls используем, чтобы заполнить XXX
в `pgbench ... -f group1.sql@XXX`

Агрегированные отчёты — K001 и K002, узнаём много интересного

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

Calls	Total time	Rows	shared_blks_hit	shared_blks_read	shared_blks_dirtied	share
143,004 376.86/sec 1.00/call 100.00%	563,081.09 ms 1.483s/sec 3ms/call 100.00%	308,255 812.34/sec 2.16/call 100.00%	120,731,634 blks 318.17K blks/sec 844.25 blks/call 100.00%	780,781 blks 2.06K blks/sec 5.46 blks/call 100.00%	21,600 blks 56.92 blks/sec 0.15 blks/call 100.00%	118 bl 0.31 b 0.00 b 100.0

K001 – характеристика всего «потока»

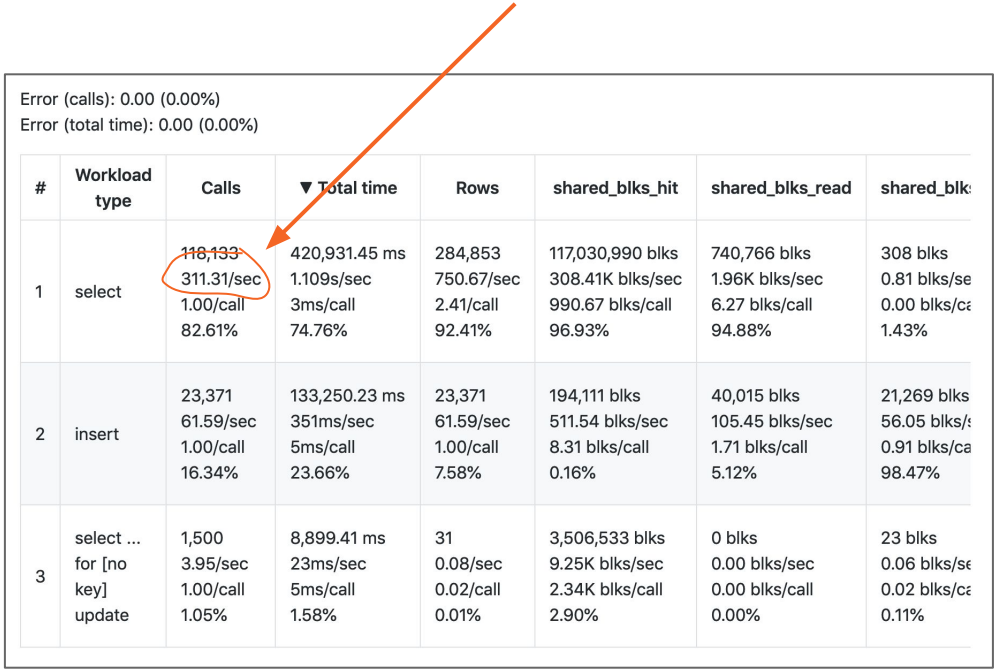
Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

#	Workload type	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blk
1	select	118,133 311.31/sec 1.00/call 82.61%	420,931.45 ms 1.109s/sec 3ms/call 74.76%	284,853 750.67/sec 2.41/call 92.41%	117,030,990 blks 308.41K blks/sec 990.67 blks/call 96.93%	740,766 blks 1.96K blks/sec 6.27 blks/call 94.88%	308 blks 0.81 blks/se 0.00 blks/ca 1.43%
2	insert	23,371 61.59/sec 1.00/call 16.34%	133,250.23 ms 351ms/sec 5ms/call 23.66%	23,371 61.59/sec 1.00/call 7.58%	194,111 blks 511.54 blks/sec 8.31 blks/call 0.16%	40,015 blks 105.45 blks/sec 1.71 blks/call 5.12%	21,269 blks 56.05 blks/s 0.91 blks/ca 98.47%
3	select ... for [no key] update	1,500 3.95/sec 1.00/call 1.05%	8,899.41 ms 23ms/sec 5ms/call 1.58%	31 0.08/sec 0.02/call 0.01%	3,506,533 blks 9.25K blks/sec 2.34K blks/call 2.90%	0 blks 0.00 blks/sec 0.00 blks/call 0.00%	23 blks 0.06 blks/se 0.02 blks/ca 0.11%

K002 – «классы» запросов:
`SELECT`, `SELECT .. FOR UPDATE`,
`INSERT` и т.д.

Цель оптимизации: на что смотреть — throughput или latency?

Большинство опубликованных бенчмарков (особенно pgbench) ориентированы на **max throughput** – максимальное количество транзакций в секунду, max TPS



Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

#	Workload type	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blk
1	select	118,133 311.31/sec 1.00/call 82.61%	420,931.45 ms 1.109s/sec 3ms/call 74.76%	284,853 750.67/sec 2.41/call 92.41%	117,030,990 blks 308.41K blks/sec 990.67 blks/call 96.93%	740,766 blks 1.96K blks/sec 6.27 blks/call 94.88%	308 blks 0.81 blks/se 0.00 blks/ca 1.43%
2	insert	23,371 61.59/sec 1.00/call 16.34%	133,250.23 ms 351ms/sec 5ms/call 23.66%	23,371 61.59/sec 1.00/call 7.58%	194,111 blks 511.54 blks/sec 8.31 blks/call 0.16%	40,015 blks 105.45 blks/sec 1.71 blks/call 5.12%	21,269 blks 56.05 blks/s 0.91 blks/ca 98.47%
3	select ... for [no key] update	1,500 3.95/sec 1.00/call 1.05%	8,899.41 ms 23ms/sec 5ms/call 1.58%	31 0.08/sec 0.02/call 0.01%	3,506,533 blks 9.25K blks/sec 2.34K blks/call 2.90%	0 blks 0.00 blks/sec 0.00 blks/call 0.00%	23 blks 0.06 blks/se 0.02 blks/ca 0.11%

Цель оптимизации: на что смотреть — throughput или latency?

Большинство опубликованных бенчмарков (особенно pgbench) ориентированы на **max throughput** – максимальное количество транзакций в секунду, max TPS

Error (calls): 0.00 (0.00%)
Error (total time): 0.00 (0.00%)

#	Workload type	Calls	▼ Total time	Rows	shared_blks_hit	shared_blks_read	shared_blk
1	select	118,133 311.31/sec 1.00/call 82.61%	420,931.45 ms 1.109s/sec 3ms/call 74.76%	284,853 750.67/sec 2.41/call 92.41%	117,030,990 blks 308.41K blks/sec 990.67 blks/call 96.93%	740,766 blks 1.96K blks/sec 6.27 blks/call 94.88%	308 blks 0.81 blks/se 0.00 blks/ca 1.43%
2	insert	23,371 61.59/sec 1.00/call 16.34%	133,250.23 ms 351ms/sec 5ms/call 23.66%	23,371 61.59/sec 1.00/call 7.58%	194,111 blks 511.54 blks/sec 8.31 blks/call 0.16%	40,015 blks 105.45 blks/sec 1.71 blks/call 5.12%	21,269 blks 56.05 blks/s 0.91 blks/ca 98.47%
3	select ... for [no key] update	1,500 3.95/sec 1.00/call 1.05%	8,899.41 ms 23ms/sec 5ms/call 1.58%	31 0.08/sec 0.02/call 0.01%	3,506,533 blks 9.25K blks/sec 2.34K blks/call 2.90%	0 blks 0.00 blks/sec 0.00 blks/call 0.00%	23 blks 0.06 blks/se 0.02 blks/ca 0.11%

В «обычной» жизни боевые БД не «ездят» на максимальной скорости (как и автомобили на обычных дорогах).

Альтернативный подход: находить **min latency** – минимальное среднее время отклика

Резюме и советы

- Создайте Database Lab!
  - с использованием Nancy CLI (или соответствующих идей)
  - “Staging on demand” — временные экземпляры, по запросу
  - регулярно готовьте копии боевых БД
  - научитесь эмулировать нагрузку
    (по логам / pgreplay или «крафтовая нагрузка» с pgbench)
- На любой любое изменение, на любой вопрос — эксперименты над БД
- Прицел на min latency
- Используйте исходники Постгреса, используйте поиск
  (https://github.com/postgres/postgres или http://gitlab.com/postgres/postgres)
- Проверяйте здоровье своих БД регулярно (postgres-checkup)

Спасибо!

Nikolay Samokhvalov
nik@postgres.ai
twitter: @postgresmen

Nancy CLI:

https://gitlab.com/postgres-ai-team/nancy

postgres-checkup:

https://gitlab.com/postgres-ai-team/postgres-checkup

Кое-что из «режиссёрской версии»

Nancy real-world examples: `seq_page_cost`

GitLab.com: `random_page_cost = 1.5` and `seq_page_cost = 1`

Decision was made to switch to `seq_page_cost = 4`

DB experiments with Nancy CLI were made to check was this decision good in terms of performance.

Results:
https://gitlab.com/gitlab-com/gl-infra/infrastructure/issues/4835#note_106669373

– in general, SQL performance **improved ~40%**

WIP here, it is an open question, why is it so.

Nancy real-world examples: educate yourself

PostgreSQL Documentation “19.5. Write Ahead Log”

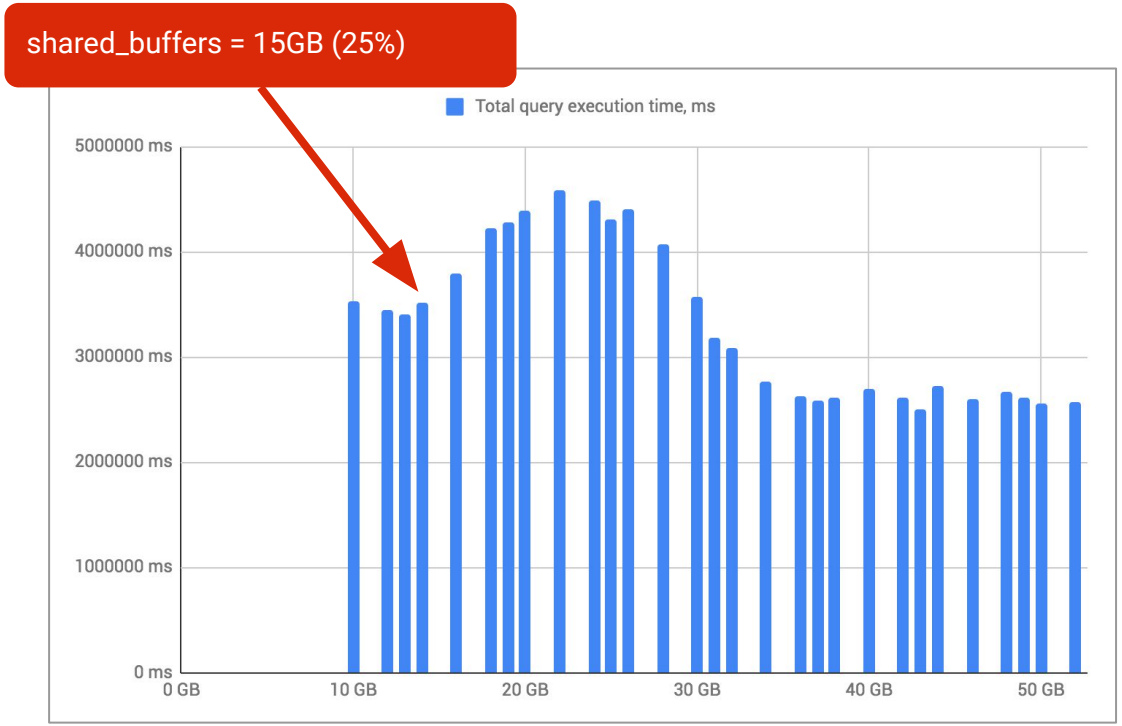
https://www.postgresql.org/docs/current/static/runtime-config-wal.html

`wal_level` (`enum`)

`wal_level` determines how much information is written to the WAL. The default value is `replica`, which writes enough data to support WAL archiving and replication, including running read-only queries on a standby server. `minimal` removes all logging except the information required to recover from a crash or immediate shutdown. Finally, `logical` adds information necessary to support logical decoding. Each level includes the information logged at all lower levels. This parameter can only be set at server start.

In `logical` level, the same information is logged as with `replica`, plus information needed to allow extracting logical change sets from the WAL. Using a level of `logical` will increase the WAL volume, particularly if many tables are configured for `REPLICA IDENTITY FULL` and many `UPDATE` and `DELETE` statements are executed.

Nancy real-world examples: `shared_buffers`



postila.ru,

Real workload (5 min),

61 GB RAM (ec2 i3.2xlarge),

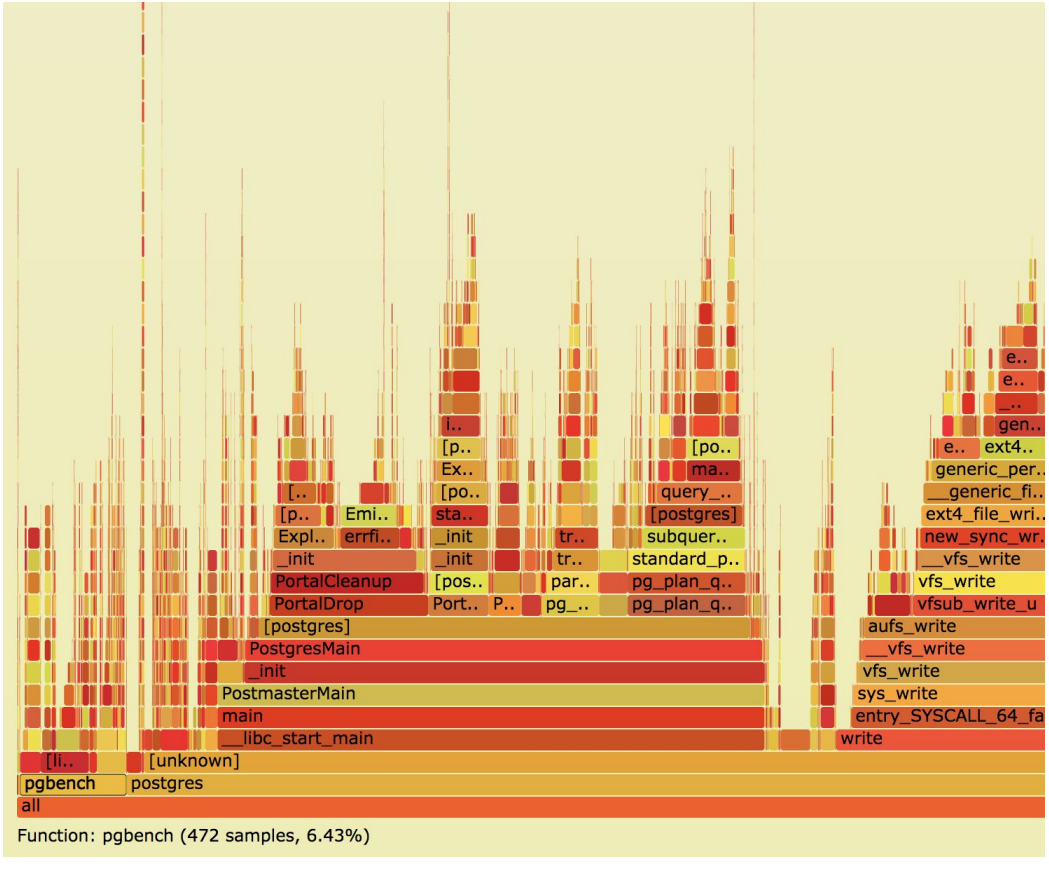
DB size: ~500GB

Various **`shared_buffers`** values

If we go from 25% o higher values (~80%), we improve SQL performance ~50%

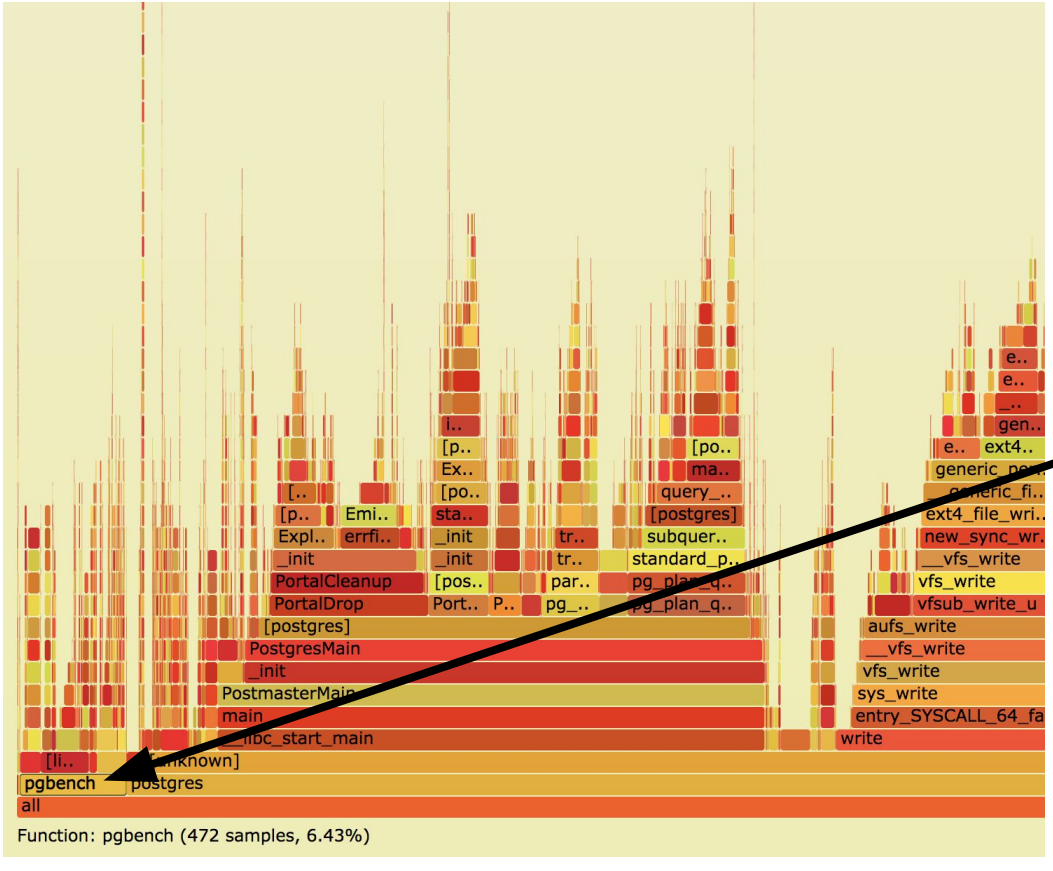
pgbench/pgreplay — на той же машине, что и Postgres
или обязательно отдельно?

pgbench/pgreplay — на той же машине, что и Postgres
или обязательно отдельно?



FlameGraphs/perf
(thanks Victor Yagofarov!
https://github.com/postgres-ai/nancy/pull/159)

pgbench/pgreplay — на той же машине, что и Postgres
или обязательно отдельно?



«Вклад» pgbench всего 6%

FlameGraphs/perf
(thanks Victor Yagofarov!
https://github.com/postgres-ai/nancy/pull/159)

Nancy real-world examples: educate yourself

PostgreSQL Documentation “19.5. Write Ahead Log”

https://www.postgresql.org/docs/current/static/runtime-config-wal.html

Just conduct DB experiment with Nancy CLI, use --keep-alive 3600 and compare!

`wal_level` (`enum`)

`wal_level` determines how much in
`replica` , which writes enough data t
read-only queries on a standby server. `minimal` removes all logging except the information required to recover from a crash or immediate shutdown. Finally, `logical` adds information necessary to support logical decoding. Each level includes the information logged at all lower levels. This parameter can only be set at server start.

In `logical` level, the same information is logged as with `replica` , plus information needed to allow extracting logical change sets from the WAL. Using a level of `logical` will increase the WAL volume, particularly if many tables are configured for `REPLICA IDENTITY FULL` and many `UPDATE` and `DELETE` statements are executed.

Главное:

- Эксперименты БД — переход от «чёрной магии» к промышленным методам и решениям, основанным *на данных*
- Staging DB — как можно ближе к production
- Лучше иметь много “staging DB”. Ещё лучше — создавать по запросу
- Используйте готовые open source решения для того, чтобы знать о своей БД и нагрузке как можно больше

Из чего состоит эксперимент над БД

Из чего состоит эксперимент над БД

Входящие:

1. Среда

   “железо”, ОС, ФС,
   версия Postgres, конфигурация

Из чего состоит эксперимент над БД

Входящие:

1. Среда

   “железо”, ОС, ФС,
   версия Postgres, конфигурация

2. Объект

   Некоторая БД (например, “клон прода”)

Из чего состоит эксперимент над БД

Входящие:

1. Среда
   “железо”, ОС, ФС,
   версия Postgres, конфигурация
2. Объект
   Некоторая БД (например, “клон прода”)
3. Нагрузка
   Некоторый набор SQL-запросов

Из чего состоит эксперимент над БД

Входящие:

1. Среда
   - “железо”, ОС, ФС,
   - версия Postgres, конфигурация
2. Объект
   - Некоторая БД (например, “клон прода”)
3. Нагрузка
   - Некоторый набор SQL-запросов
4. Изменение (может быть несколько значений)
   - Некоторое изменение конфига Postgres
   - или, например, новый индекс

Из чего состоит эксперимент над БД

Входящие:

1. Среда
   “железо”, ОС, ФС,
   версия Postgres, конфигурация
2. Объект
   Некоторая БД (например, “клон прода”)
3. Нагрузка
   Некоторый набор SQL-запросов
4. Изменение (может быть несколько значений)
   Некоторое изменение конфига Postgres
   или, например, новый индекс

Результат:

1. Summary
   лучше или хуже, в целом?

Из чего состоит эксперимент над БД

Входящие:

1. Среда
   “железо”, ОС, ФС,
   версия Postgres, конфигурация
2. Объект
   Некоторая БД (например, “клон прода”)
3. Нагрузка
   Некоторый набор SQL-запросов
4. Изменение (может быть несколько значений)
   Некоторое изменение конфига Postgres
   или, например, новый индекс

Результат:

1. Summary
   лучше или хуже, в целом?
2. Артифакты
   любые полезные подробности

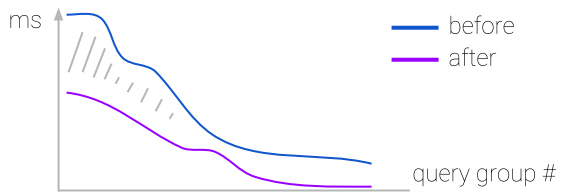
Из чего состоит эксперимент над БД

Входящие:

1. Среда
   “железо”, ОС, ФС,
   версия Postgres, конфигурация
2. Объект
   Некоторая БД (например, “клон прода”)
3. Нагрузка
   Некоторый набор SQL-запросов
4. Изменение (может быть несколько значений)
   Некоторое изменение конфига Postgres
   или, например, новый индекс

Результат:

1. Summary
   лучше или хуже, в целом?
2. Артифакты
   любые полезные подробности
3. Подробный анализ SQL-запросов
   каждая группа: лучше или хуже?
   + гистограммы:



Возможно ли посредством существующих решений?

- Docker
- pgreplay
- pg_stat_***
- auto_explain
- pgBadger (with JSON output)
- AWS EC2 spot instances

— Необходимые “строительные блоки” уже существуют

Возможно ли посредством существующих решений?

- Docker
- pgreplay
- pg_stat_***
- auto_explain
- pgBadger (with JSON output)
- AWS EC2 spot instances

— Необходимые “строительные блоки” уже существуют

Nancy CLI: эти (и не только) блоки, интегрированные в единое решение
https://github.com/postgres-ai/nancy

DIY automated pipeline for DB experiments and optimization

How to automate database optimization using ecosystem tools and AWS?

Analyze:

- pg_stat_statements
- auto_explan
- pgBadger to parse logs, use JSON output
- pg_query to group queries better
- pg_stat_kcache to analyze FS-level ops

pgBadger:

- Grouping queries can be implemented better (see pg_query)
- Makes all queries lower cased (hurts `"camelCased"` names)*
- Doesn't really support plans (auto_explain)*

*) Fixed/improved in pgBadger 10.0

Configuration:

- annotated.conf, pgtune, pgconfigurator, postgresqlco.nf
- ottertune

Suggested indexes (internal "what-if" API w/o actual execution)

- (useful: pgHero, POWA, HypoPG, dexter, plantuner)

pgreplay and **pgBadger** are not friends, require different log formats

Conduct experiments:

- pgreplay to replay logs (different `log_line_prefix`, you need to handle it)
- EC2 spot instances

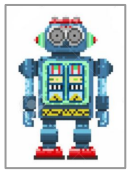
Machine learning

- MADlib

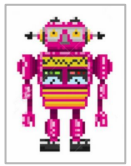
Postgres.ai — artificial DBA/DBRE assistants

AI-based cloud-friendly platform to automate database administration

Steve
AI-based expert in **capacity planning** and **database tuning**



Joe
AI-based expert in **query optimization** and Postgres **indexes**



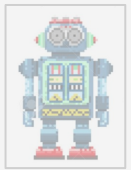
Nancy
AI-based expert in database experiments. Conducts **experiments** and presents results to human and artificial DBAs

https://Postgres.ai

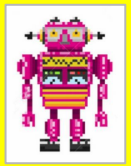
Postgres.ai — artificial DBA/DBRE assistants

AI-based cloud-friendly platform to automate database administration

Steve
AI-based expert in **capacity planning** and **database tuning**



Joe
AI-based expert in **query optimization** and Postgres **indexes**



Nancy
AI-based expert in database experiments. Conducts **experiments** and presents results to human and artificial DBAs

https://Postgres.ai

Что внутри docker-контейнера Nancy

Исходники: https://github.com/postgres-ai/nancy/tree/master/docker
Образ: https://hub.docker.com/r/postgresmen/postgres-nancy

Внутри:

- Ubuntu 16.04
- Postgres (9.6, 10, 11)
- postgres_dba на случай ручного “дебаггинга” https://github.com/NikolayS/postgres_dba
- `log_min_duration_statement = 0`
- pg_stat_statements включены, с `track_io_timing = on`
- auto_explain опционально
- pgreplay
- pgBadger
- FlameGraph (perf cpu) https://github.com/postgres-ai/nancy/pull/159

- plain text pg_dump
  - restoration is very slow (1 vcpu utilized)
  - “logical” – physical structure is lost (cannot experiment with bloat, etc)
  - small (if compressed)
  - “snapshot” only
- pg_dump with either -Fd (“directory”) or -Fc (“custom”):
  - restoration is faster (multiple vCPUs, -j option)
  - “logical” (again: bloat, physical layout is “lost”)
  - small (because compressed)
  - “snapshot” only
- pg_basebackup + WALs, point-in-time recovery (PITR), possibly with help from WAL-E, WAL-G, pgBackRest
  - less reliable, sometimes there issues (especially if 3rd party tools involved - e.g. WAL-E & WAL-G don’t support tablespaces, there are bugs sometimes, etc)
  - “physical”: bloat and physical structure is preserved
  - not small – ~ size of the DB
  - can “walk in time” (PITR)
  - requires warm-up procedure (data is not in the memory!)
- AWS RDS: create a replica + promote it
  - no Spots :-/
  - Lazy Load is tricky (it looks like the DB is there but it’s very slow – warm-up is needed)
- Snapshots
- Ideas for serialization
  - Stop Postgres / rsync “back” or re-copy locally on NVMe / start Postgres

- Prepare the EC2 instance(s) in advance and keep it
- Prepare EBS volume(s) only (perhaps, using an instance of the different type) and keep it ready. When attached to the new instance, do warm-up
- Resource re-usage:
    - reuse docker container
    - reuse EC2 instance
    - serialize experimental runs serialization (DDL Do/Undo; VACUUM FULL; cleanup)
- Partial database snapshots (dump/restore only needed tables)
- Filesystem snapshots to have few-second resets
  (examples: https://events.yandex.ru/lib/talks/4402/, https://heapanalytics.com/blog/engineering/testing-database-changes-right-way)

- ZFS/XFS snapshots to revert PGDATA state within seconds
- FlameGraphs (perf) – **DONE** https://github.com/postgres-ai/nancy/pull/159
- Support GCP
- More artifacts delivered: pg_stat_kcache, etc
- `nancy describe` to print the summary + top-N queries **– DONE**
- `nancy describe` to print the "diff" for 2+ reports (the summary + numbers for top-30 queries, ordered by by total time based on the 1st report) **– DONE**
- Postgres 11 **– DONE**
- `pgbench -i` for database initialization – **DONE**
- pgbench to generate multithreaded synthetic workload – **DONE**
- Workload analysis: automatically detect "N+1 SELECT" when running workload
- Better support for the serialization of experimental runs
- Better support for multiple runs https://github.com/postgres-ai/nancy/pull/97
  - interval with step **– WIP**
  - gradient descent
- Provide costs estimation (time + money) **– DONE**
- Go

Feedback/contributions welcome

https://github.com/postgres-ai/nancy

Challenge: security issues

Problem: a developer doesn’t have access to production.
Nancy works with production data/workload.
What about permissions and data protection?

Possible solutions:

- Option 1: allow using Nancy CLI only to those who already have access production (DBAs, SREs, DBREs)
- Option 2: obfuscate data when preparing a DB clone (no universal solution yet, TODO)
- Option 3: allow access only to GUI, hide/obfuscate parameters (TODO)

Challenge: reliable results

Issues:

1. Single runs is not enough (fluctuations) – must repeat
2. “Before”/”after” runs on 2 different machines / EC2 nodes – “not fair” comparison (defective hardware, throttling)

Solutions (ideally: combination of them):

- Sequential runs
- 4+ iterations of each experimental run
- “Baseline benchmark” https://github.com/postgres-ai/nancy/issues/94