# Оптимизация high-contention write в PostgreSQL

Олег Бартунов, Александр Коротков, Иван Панченко

HighLoad++

Профессиональная конференция разработчиков высоконагруженных систем

Russian
Other world

Functions

Postgres95
1st commit
PostgreSQL
Triggers
Transaction isolation
1st Russian patch
Internationalization
WAL
MVCC
GiST
Windows
PITR
GIN
HStore
FTS
HOT
Exclusion constraints
CTE
XML
KNN GiST
SP-GiST
SSI
Streaming rep
Sync rep
IOS
JSON
FDW
Cascading rep
Lateral
Mat VIEW
JSONB
Logical decoding
BRIN
Greenplum
CitusDB unfork
RLS, Upsert
Generic WAL
Create AM
RUM
parallel exec, FDW pushdown
Parallel index scan
Partitioning
Logical rep
SQL/JSON
XTM
Sharding
VODKA

6.0
7.0
8.0
9.0
9.5
9.6
10.0

1995
2000
2005
2010
2015

HighLoad++ 2017

Postgres PROFESSIONAL

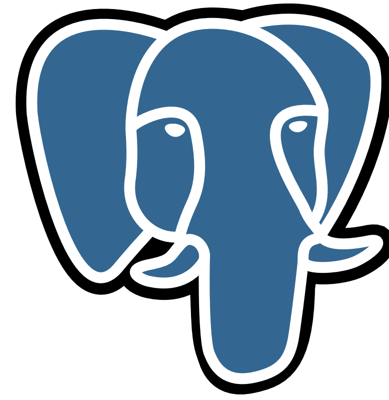
Возможности богатые, а так кто его знает ...

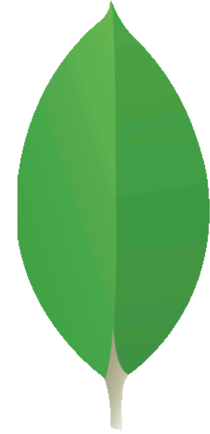
SQL/JSON 2016 ! Пора наконец померять производительность!

# Benchmarking NoSQL Postgres

- Existing benchmarks were homemade by postgres people
- People tend to believe independent and «scientific» benchmarks
  - Reproducible
  - More databases
  - Many workloads
  - Open source

# YCSB Benchmark





- Yahoo! Cloud Serving Benchmark - https://github.com/brianfrankcooper/YCSB/wiki
- De-facto standard benchmark for NoSQL databases
- Scientific paper «Benchmarking Cloud Serving Systems with YCSB» https://www.cs.duke.edu/courses/fall13/cps296.4/838-CloudPapers/ycsb.pdf
- We run YCBS for Postgres master, MongoDB 3.4.2
  - 1 server with 72 cores, 3 TB RAM, 2 TB SSD for clients
  - 1 server with 72 cores, 3 TB RAM, 2 TB SSD for database
  - 10Gbps switch

PROFESSIONAL Postgres

# YCSB Benchmark: Core workloads

- Workload A: Update heavy - a mix of 50/50 reads and writes
- Workload B: Read mostly - a 95/5 reads/write mix
- Workload C: Read only — 100% read
- Workload D: Read latest - new records are inserted, and the most recently inserted records are the most popular
- Workload E: Short ranges - short ranges of records are queried
- Workload F: Read-modify-write - the client will read a record, modify it, and write back the changes
- All (except D) workloads uses Zipfian distribution for record selections

# YCSB Benchmark: details (1)

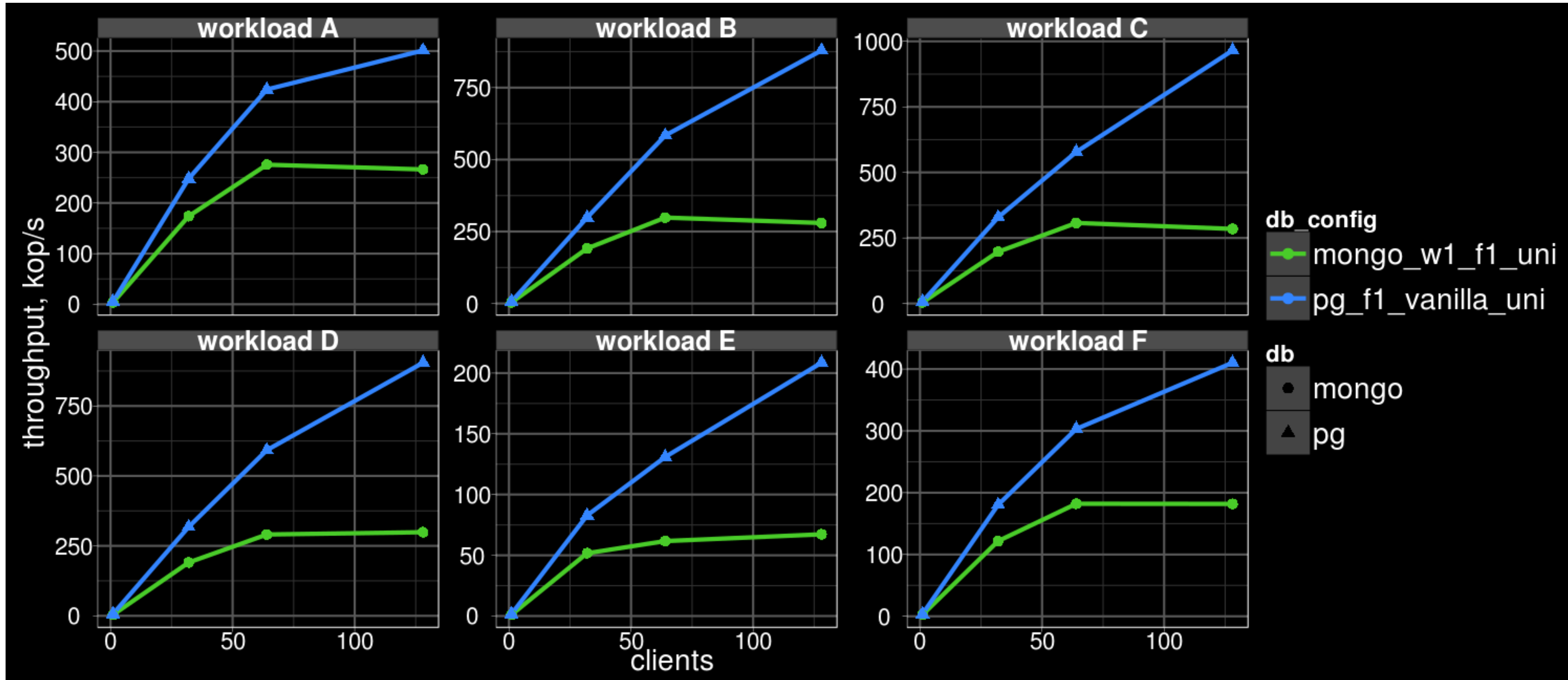
- #1: Postgres 10, synchronous commit=off
  Mongodb 3.4.2 (w1, j0) — 1 mln. Rows, #clients <= 128
- #2: Postgres 10, synchronous commit=off
  Mongodb 3.4.2 (w1, j0) — 1 mln. Rows, #clients <= 1000
- #3: Postgres 10, synchronous commit=on
  Mongodb 3.4.5 (w1, j1)
- We tested:
  - Functional btree index for jsonb, jsonbc, sqljson
  - Mongodb (wiredtiger with snappy compression)
  - Return a whole json, just one field, small range

PROFESSIONAL Postgres

HighLoad++ 2017

PROFESSIONAL Postgres

# Uniform distribution of queries: #1



# Распределение Зипфа и PostgreSQL

- Разработчики в основном пользуются pgbench. PostgreSQL – pgbench optimized DBMS.
- YCSB предлагает использовать распределение Зипфа
- В pgbench не было распределения Зипфа, исправляемся

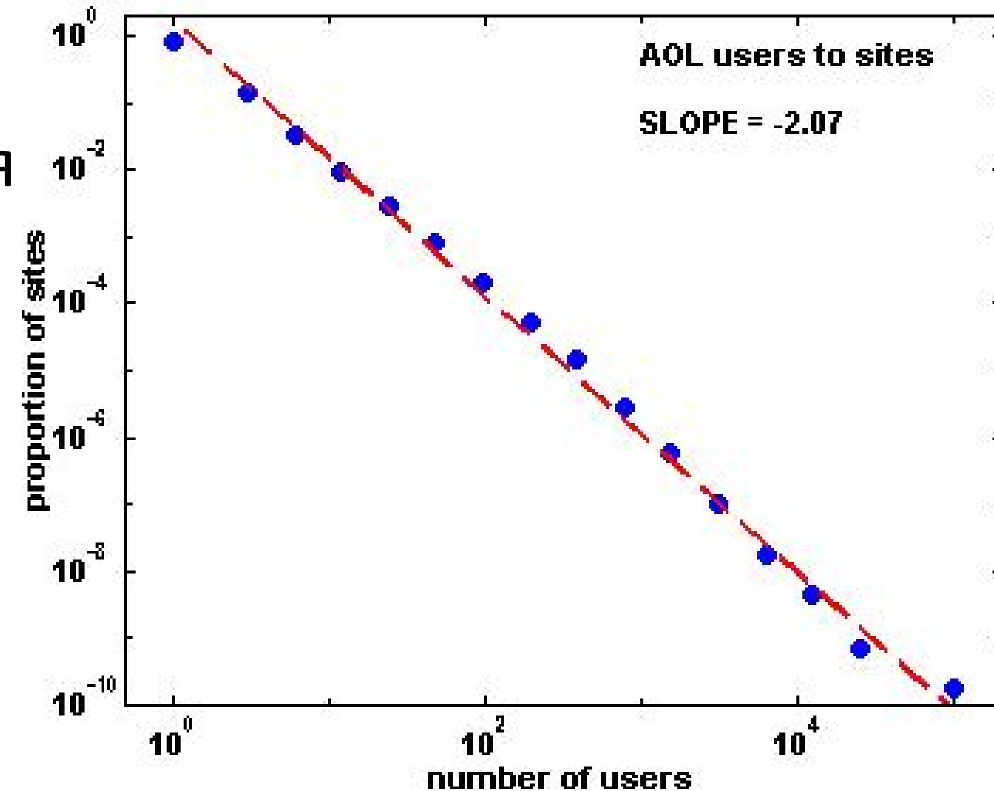
https://www.postgresql.org/message-id/flat/BF3B6F54-68C3-417A-BFAB-FB4D66F2B410@postgrespro.ru

Postgres PROFESSIONAL
HighLoad++ 2017

# Что это за распределение Зипфа

- Частота обратно пропорциональна рангу значения
- Т.е. второе по частоте значение встречается в 2 раза реже чем 1е
- Третье — в три раза реже, и т. д.

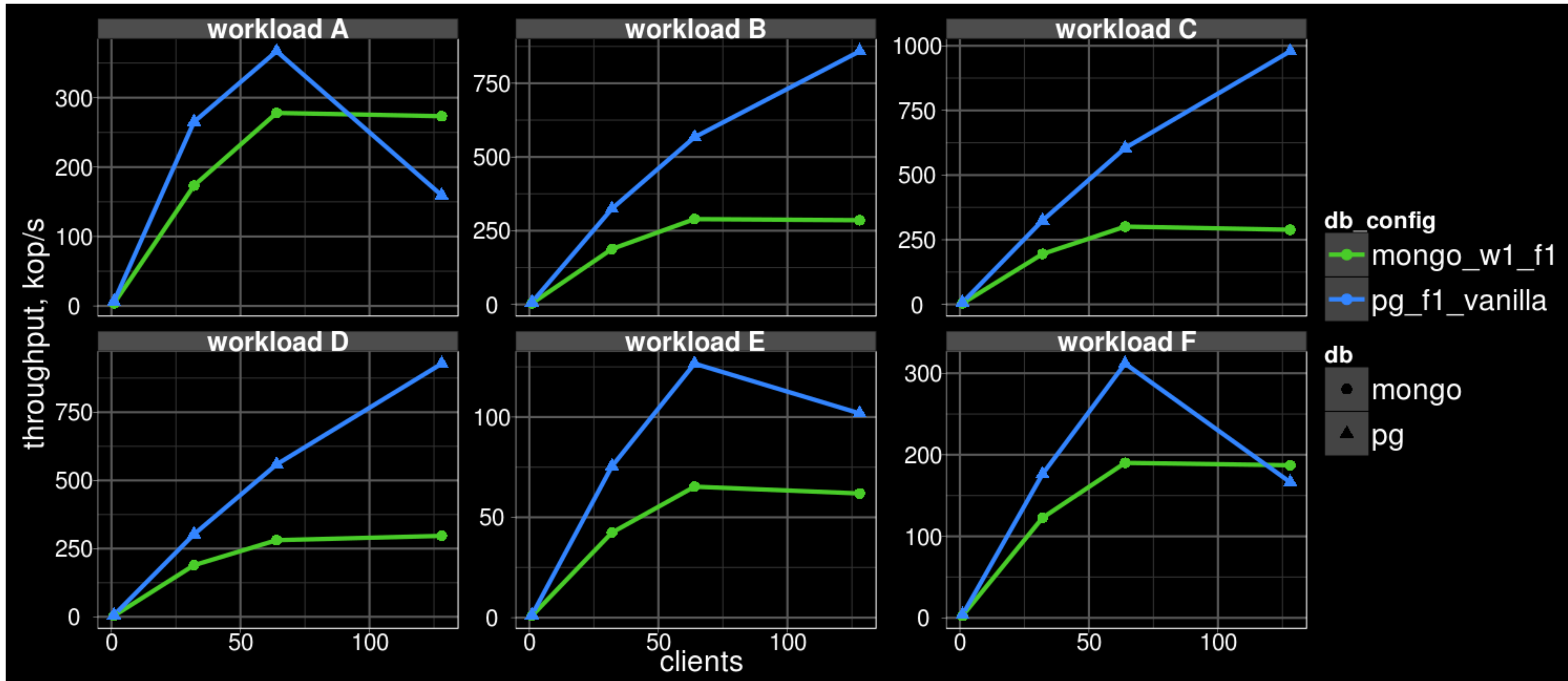
- Эмипрически выведено из анализа частотности слов
- Хорошо аппроксимирует другие жизненные ситуации




Postgres Professional
HighLoad++ 2017

PROFESSIONAL Postgres

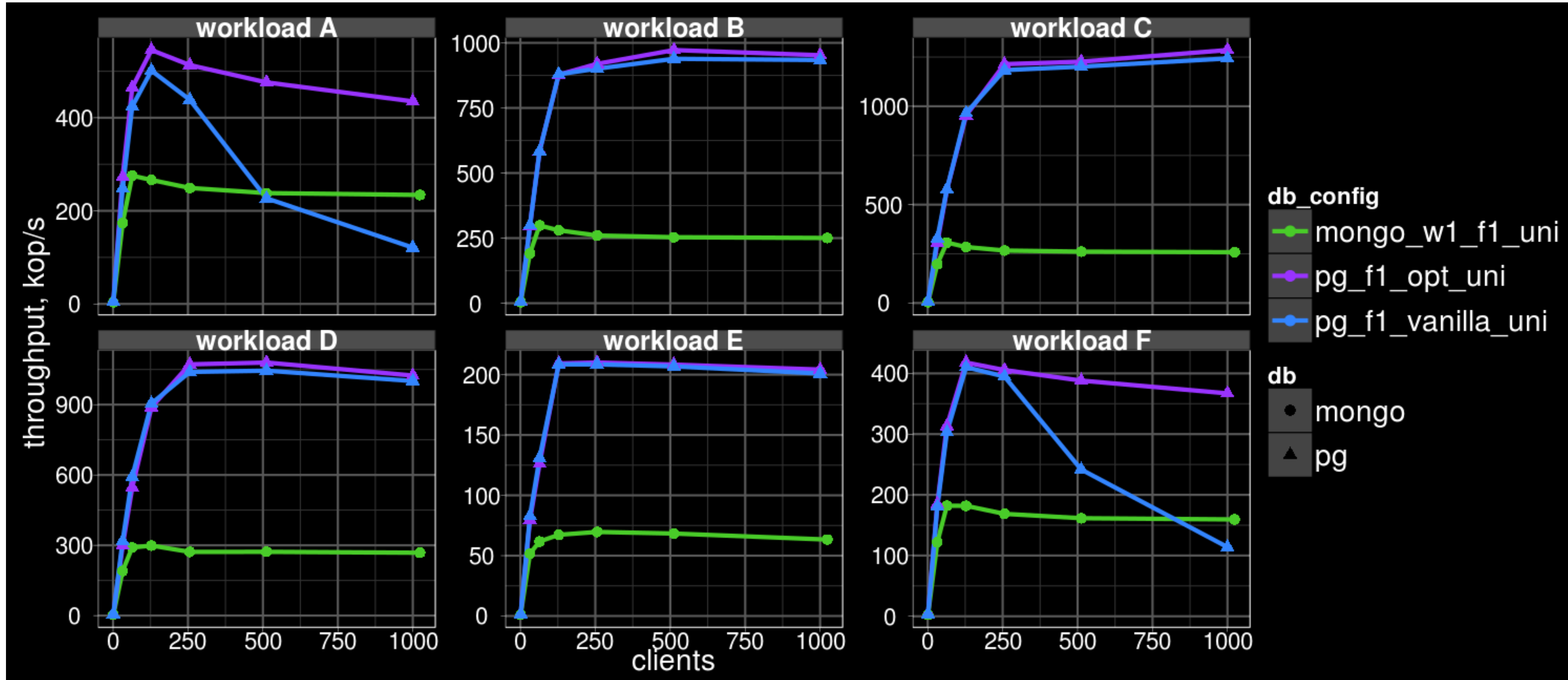
# Zipfian distributions of queries #1



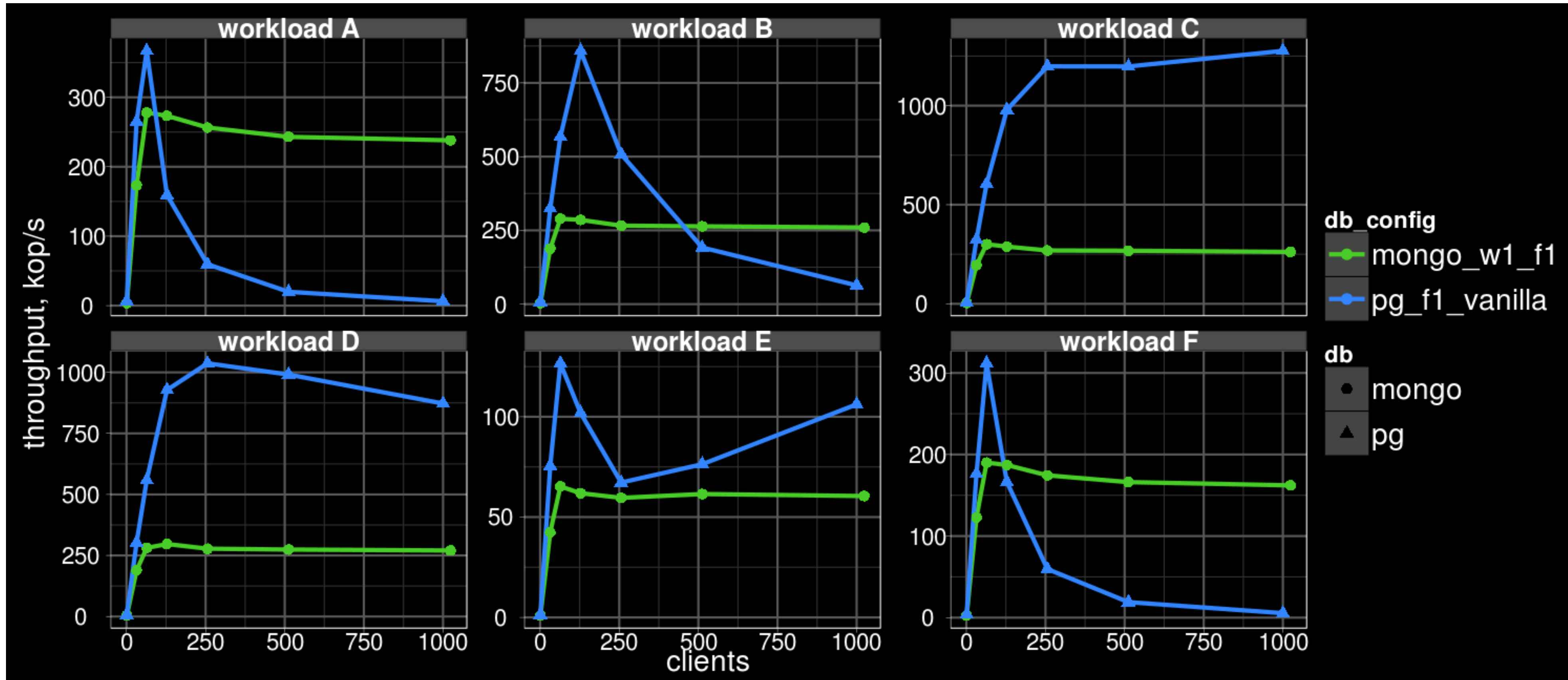

HighLoad++ 2017

PROFESSIONAL Postgres

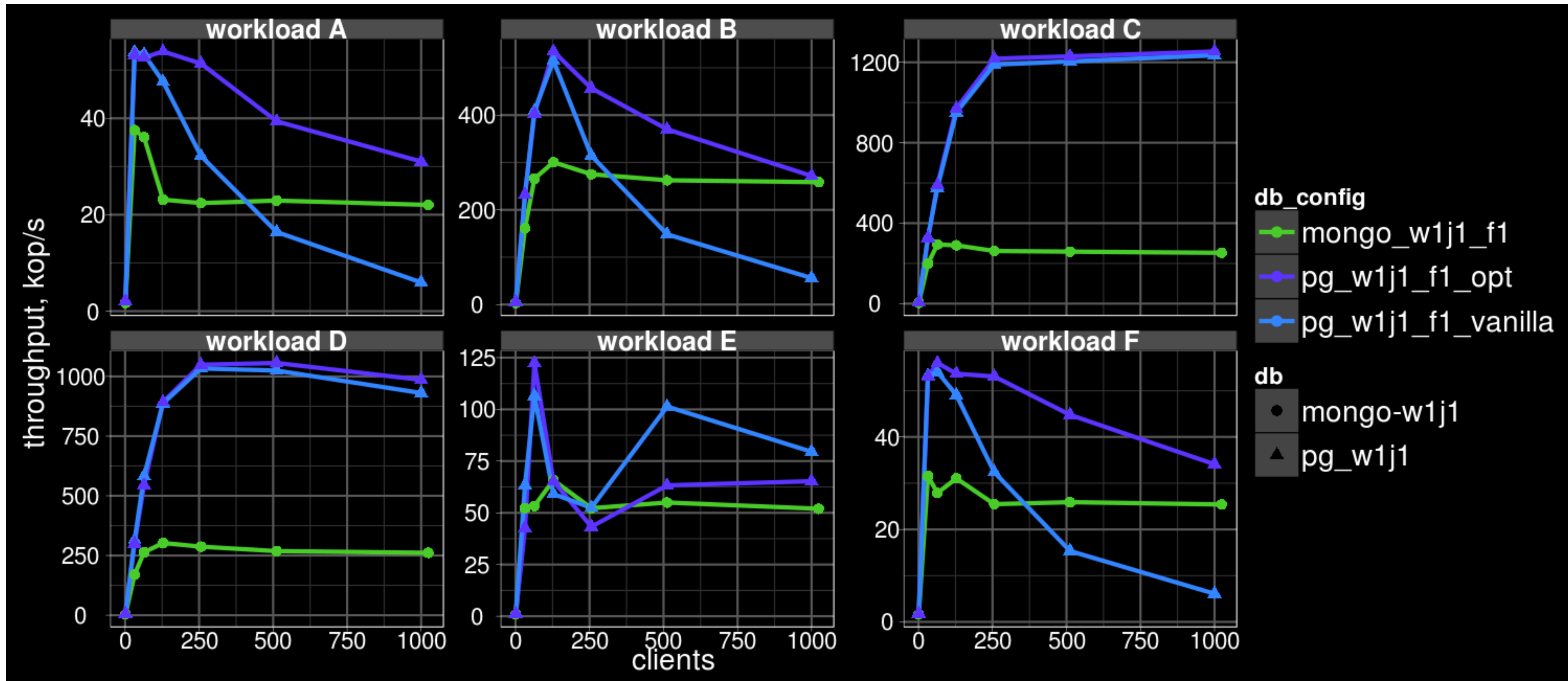
# Uniform distribution of queries: #2



# Zipfian distributions of queries: #2



# Zipfian distributions of queries: #3




PROFESSIONAL Postgres

HighLoad++ 2017

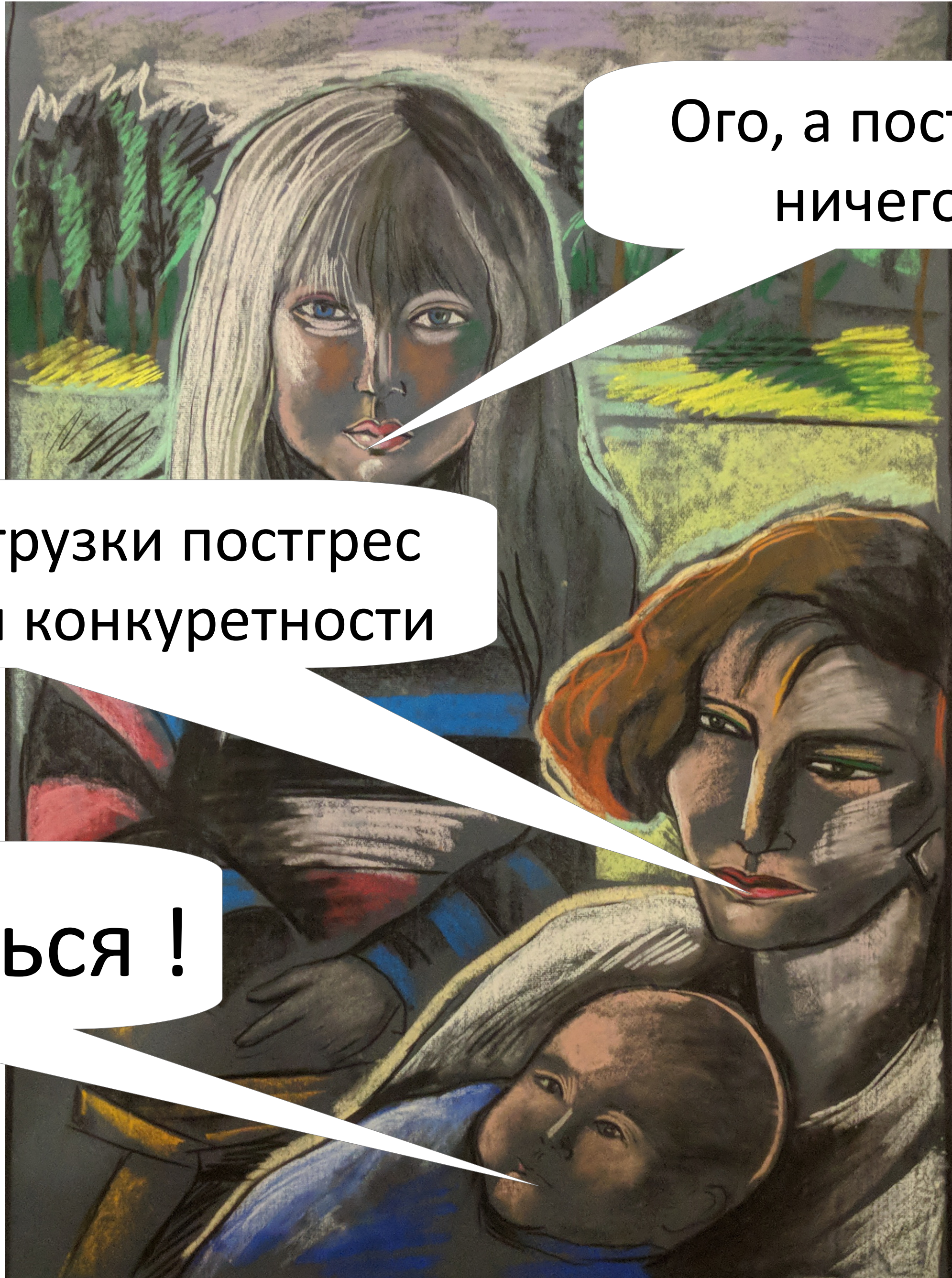
PROFESSIONAL Postgres

# Выводы

- Сочетание «synchronous_commit = off», zipfian distribution и большой конкурентности ( > 100) не очень хорошо для постгреса (обновление)
- Даже для однородного распределения при большом количестве клиентов (> 500) производительность обновления постгреса деградирует
- Персистентность «synchronous_commit = on» немного помогает постгресу конкурировать с Mongodb (j1), но не кардинально

HighLoad++ 2017

Postgres PROFESSIONAL




HighLoad++ 2017

# ACID

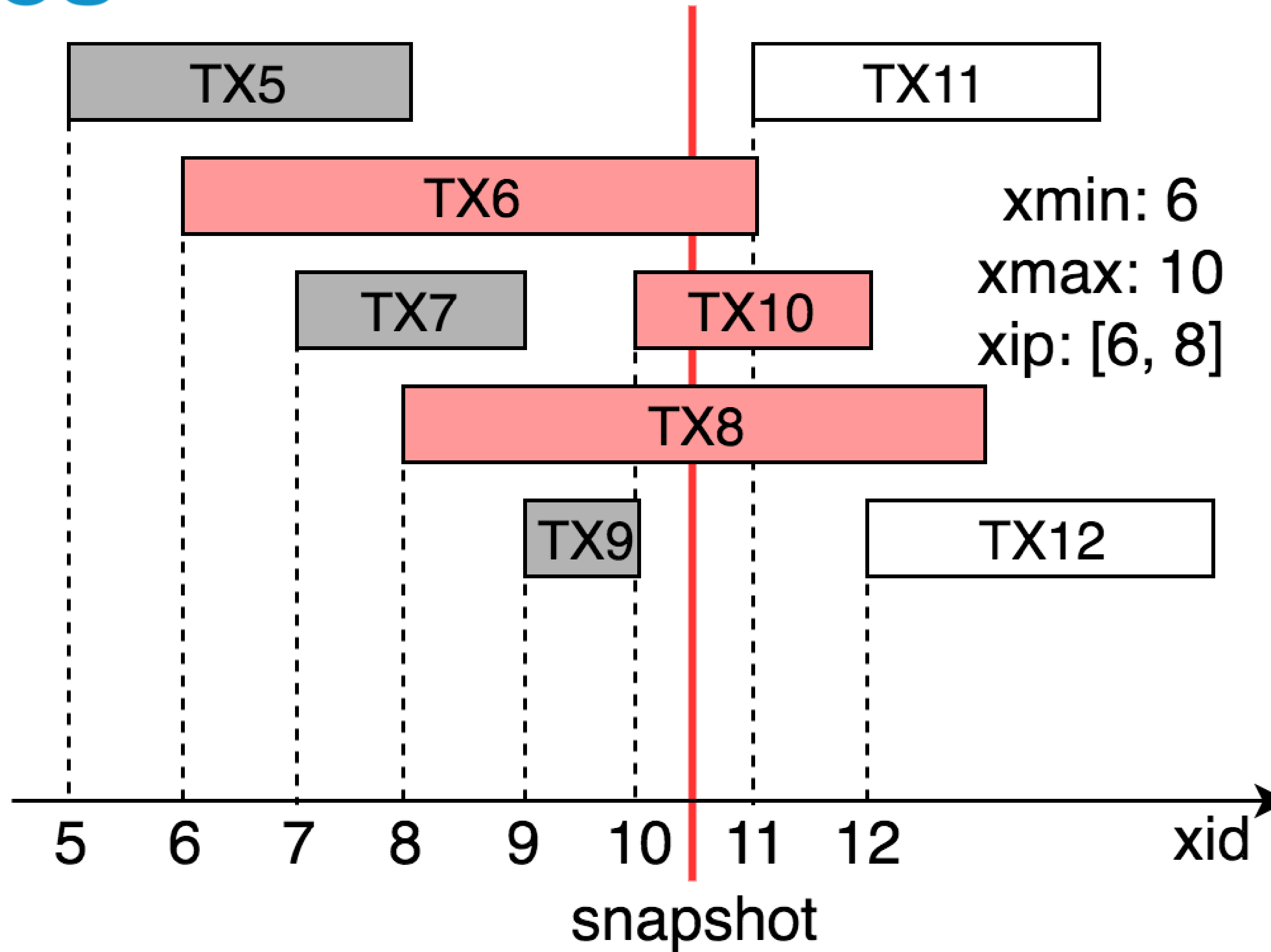
Isolation – на выполнение данной транзакции не должны оказывать влияние другие транзакции, которые работают параллельно с ней.

Для того, чтобы выполнить свойство isolation, но при этом читающие и пишущие транзакции не блокировали друг друга, был придуман MVCC. При MVCC старые версии строк сохраняются для того, чтобы читающие транзакции могли видеть их.

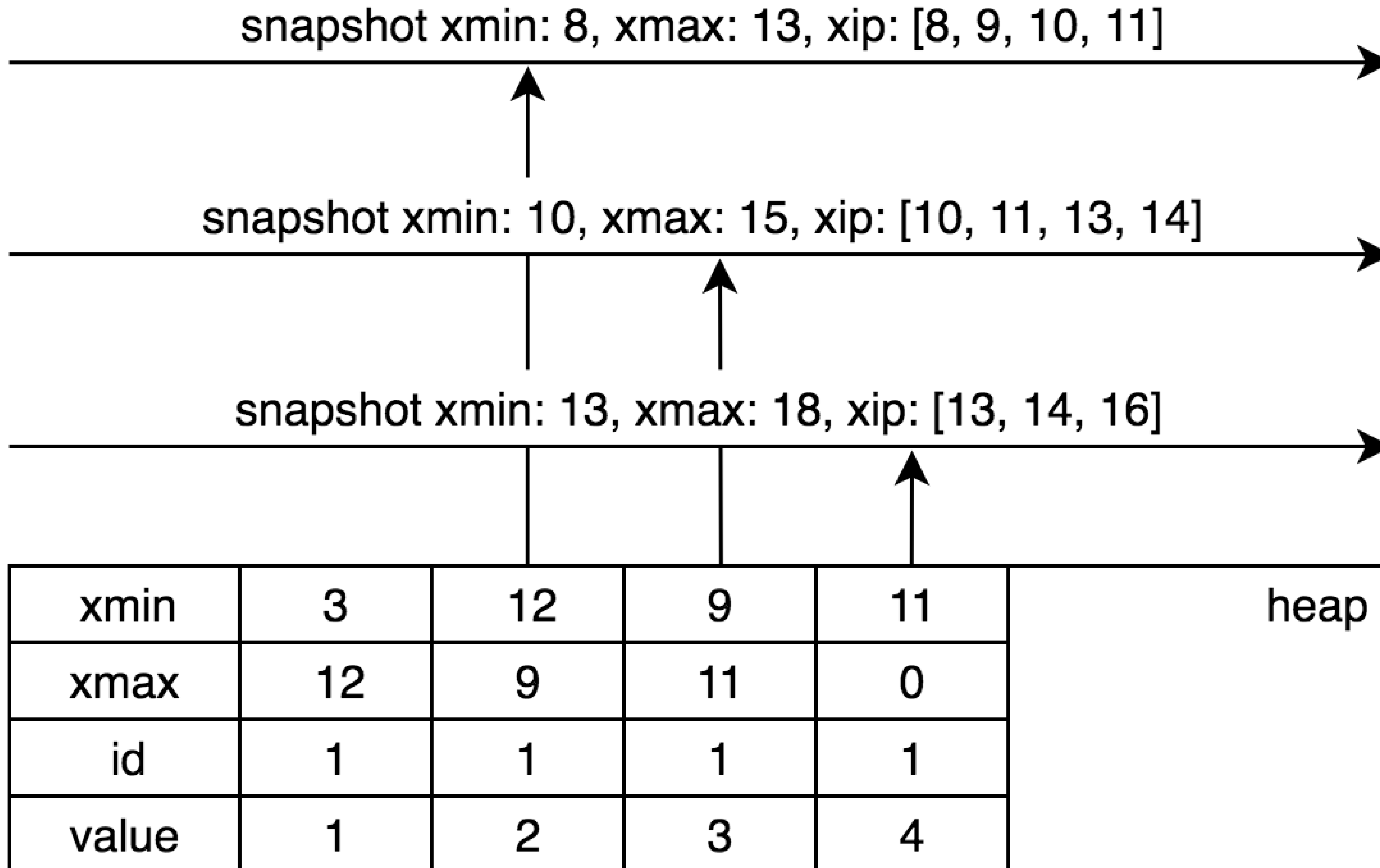
Snapshot – это состояние базы данных между двумя транзакциями. Для того, чтобы идентифицировать снапшот, нужно уметь отличать, какие транзакции в прошлом, а каким – в будущем. В PostgreSQL снапшот задаётся с помощью xmin, xmax и массива xip.

HighLoad++ 2017

# Что такое snapshot



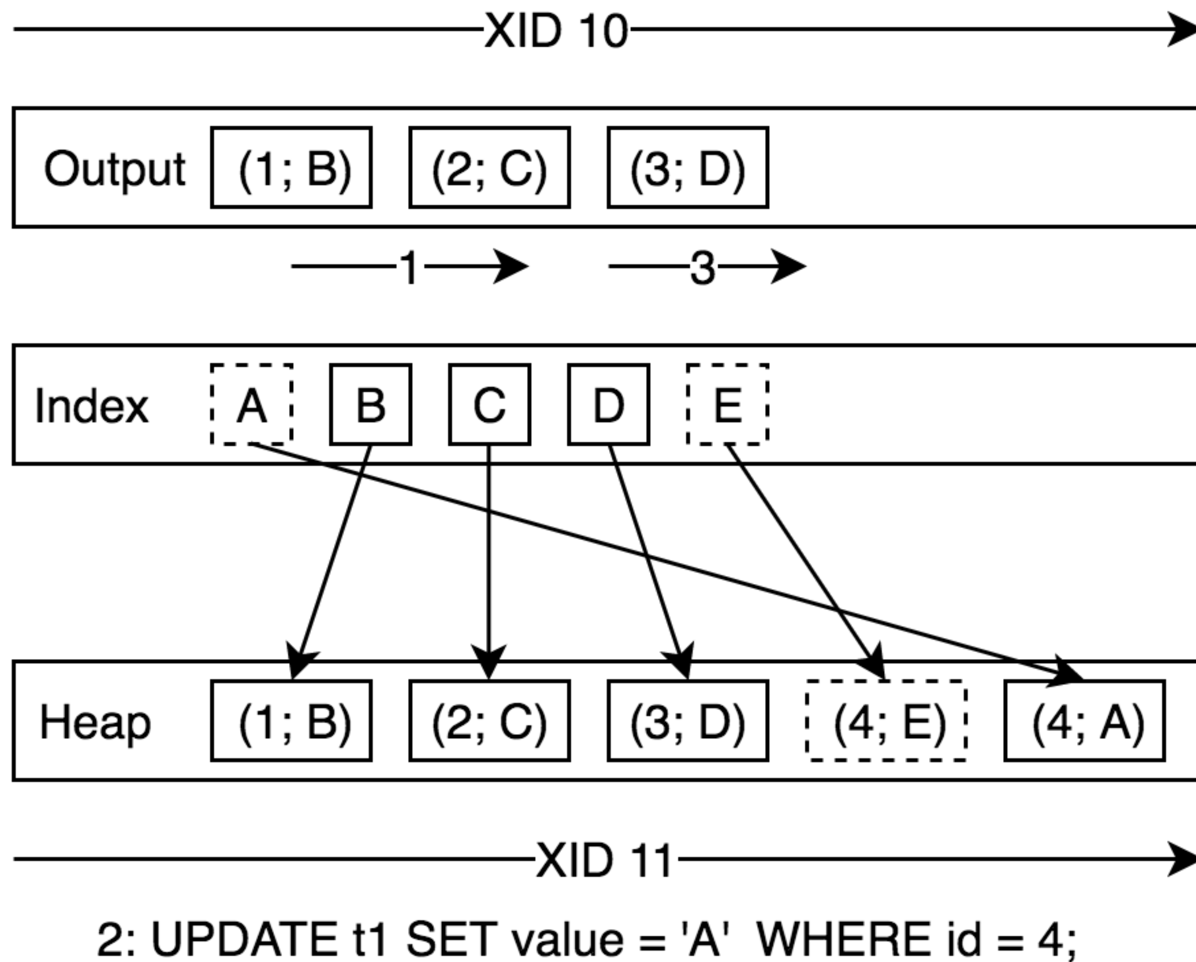
# Использование snapshot'ов




| xmin | 3 | 12 | 9 | 11 |
|---|---|---|---|---|
| xmax | 12 | 9 | 11 | 0 |
| id | 1 | 1 | 1 | 1 |
| value | 1 | 2 | 3 | 4 |

HighLoad++ 2017

# Read committed vs snapshots

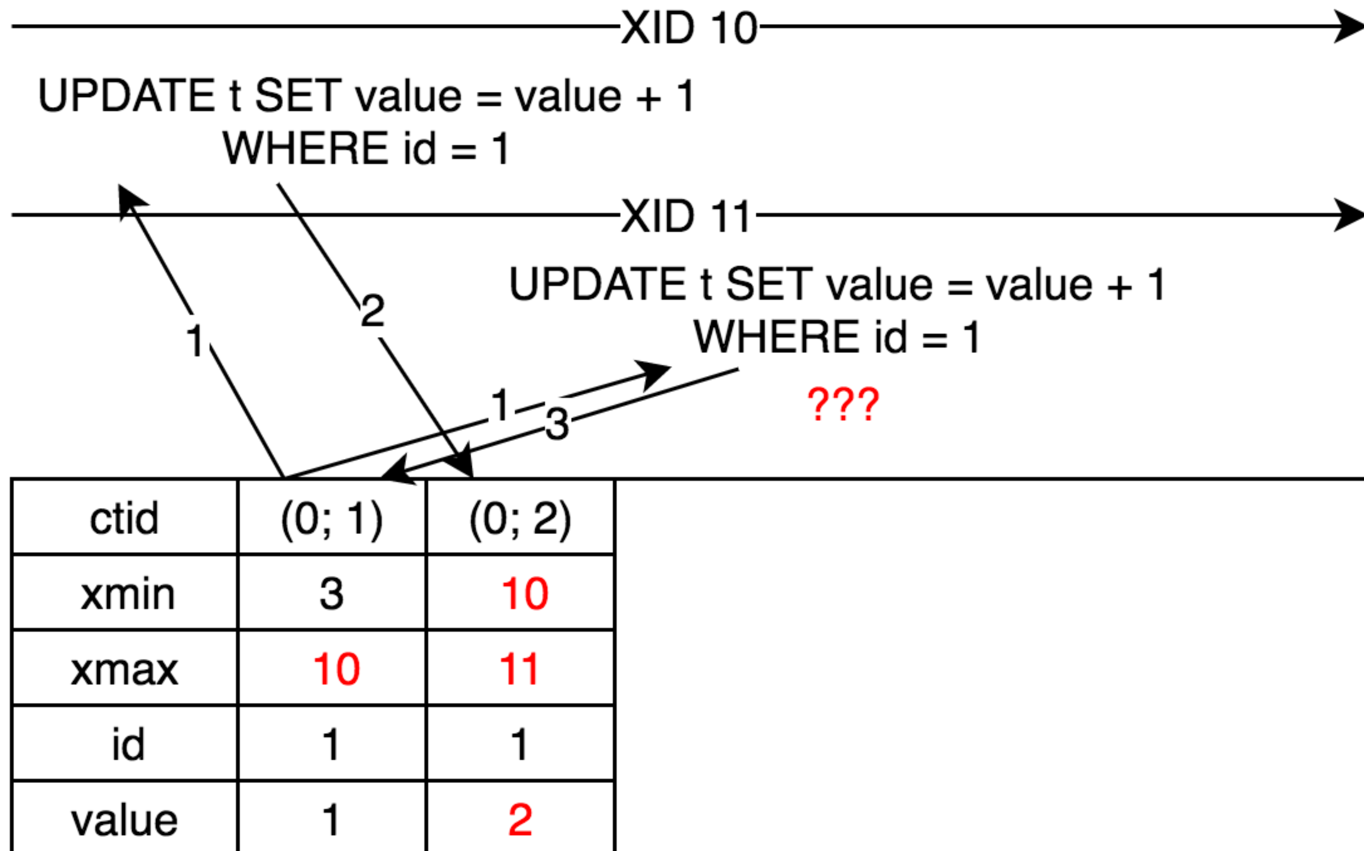



1) XID 10 читает туплы (1; B) и (2; C).
2) XID 11 обновляет тупл (4; E) на (4; A) и коммитится.
3) XID 10 читает тупл (3; D) , а (4; E) ему уже не видим, т. к. он уже был обновлён.

В итоге мы потеряли строку. Чтобы так не было нужно вначале сделать
0) XID 10 берёт снапшот для читающего запроса.

PROFESSIONAL Postgres
HighLoad++ 2017

# Contended update




| ctid | (0; 1) | (0; 2) |
|---|---|---|
| xmin | 3 | 10 |
| xmax | 10 | 11 |
| id | 1 | 1 |
| value | 1 | 2 |

1) Обе транзакции прочитали в соответствии со своими снапшотами одну и ту же версию строки ctid = (0;1). И подготовили обновлённые версии с value = 2.
2) xid 10 первая обновляет строку и добавляет версию ctid = (0;2) с value = 2.
3) xid 11 тоже пытается обновить строку ctid = (0;1), но видит что она уже обновлена... Что делать дальше?

PROFESSIONAL Postgres
HighLoad++ 2017

# ACID answer

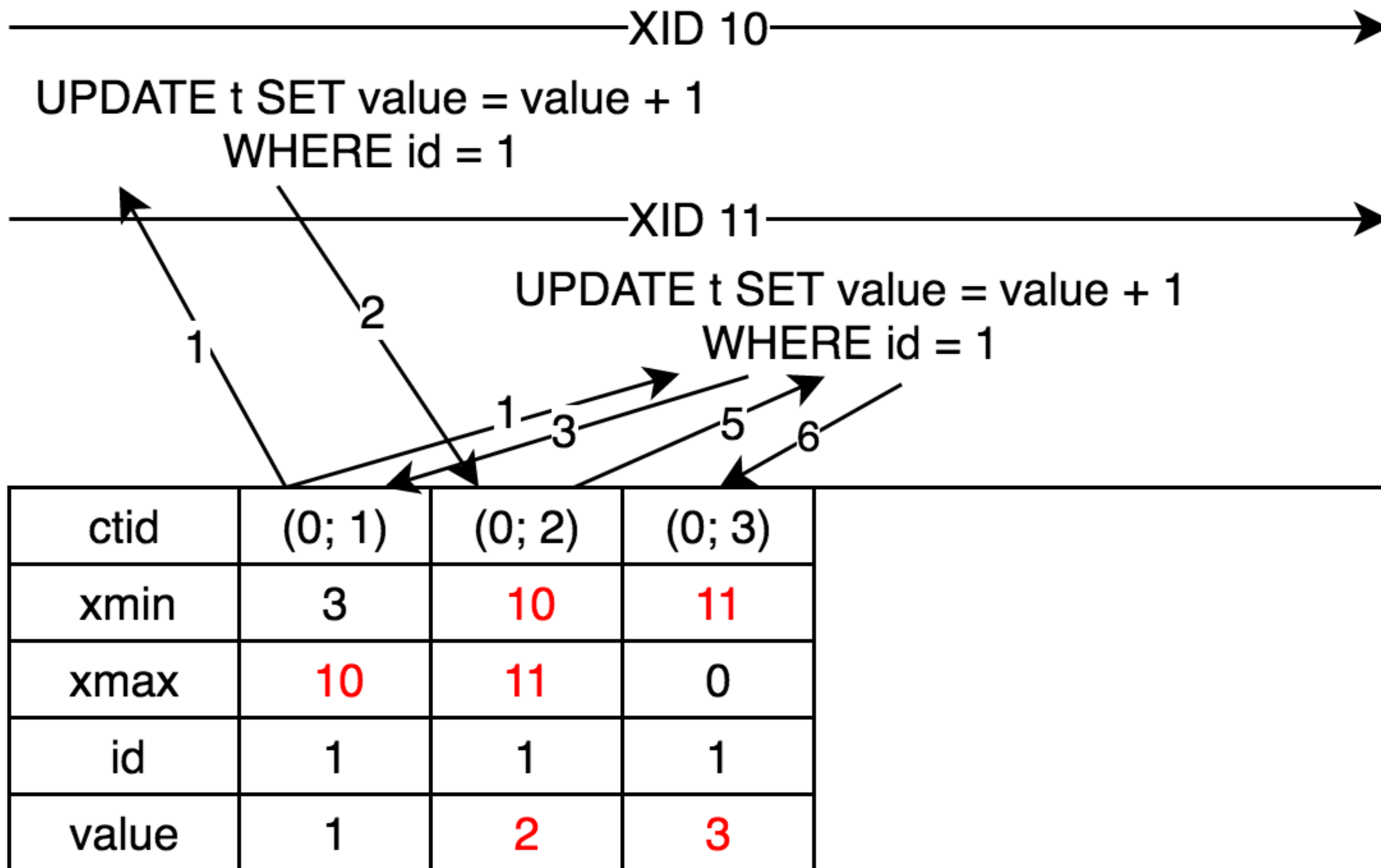
isolation level >= repeatable read

```
ERROR:  could not serialize access due to concurrent update
```

Если нам приходится обновлять строку, которую уже успела обновить другая параллельная транзакция, то это значит что её результат отразиться на результатах нашей транзакции.  Таким образом, для строгого выполнения свойства isolation нам ничего не остаётся как выдать ошибку (и попробовать нашу транзакцию ещё раз).

Postgres PROFESSIONAL

# Contended update: read committed




| ctid | (0; 1) | (0; 2) | (0; 3) |
|---|---|---|---|
| xmin | 3 | 10 | 11 |
| xmax | 10 | 11 | 0 |
| id | 1 | 1 | 1 |
| value | 1 | 2 | 3 |

1) Обе транзакции прочитали в соответствии со своими снапшотами одну и ту же версию строки ctid = (0;1). И подготовили обновлённые версии с value = 2.
2) xid 10 первая обновляет строку и добавляет версию ctid = (0;2) с value = 2.
3) xid 11 тоже пытается обновить строку ctid = (0;1), но видит что она уже обновлена и ждёт завершения xid 10.
4) xid 10 комитится.
5) xid 11 просыпается, находит последнюю версию строки ctid = (0;2) и лочит её.
6) xid 11 перевычисляет обновлённую версию строки (value = 3) и вставляет её в heap.

HighLoad++ 2017

# Как мы лочили tuple в PostgreSQL 9.4

1) Дожидаемся, пока транзакция, записанная в xmax, закончится.
2) Ищем последний tuple в цепочке update'ов.
3) Пытаемся записаться свой xid в tuple xmax.  Если кто-то уже сделал это до нас, то переходим к шагу 1.

При большом contention'е всё было совсем плохо.

Поправили в 0e5680f4.

Postgres PROFESSIONAL
HighLoad++ 2017

# Как мы лочим tuple в PostgreSQL 9.5+

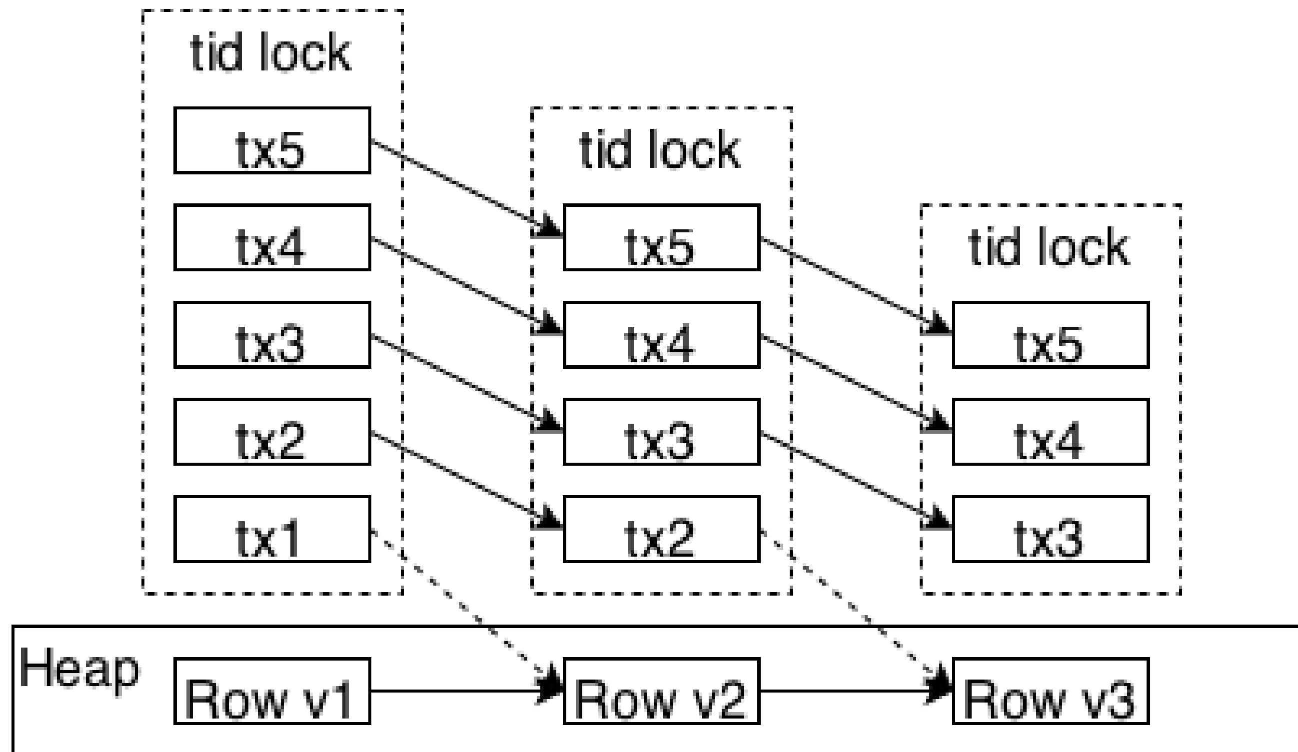
1) Берем Lock на tuple id.
2) Дожидаемся, пока транзакция, записанная в xmax, закончится.
3) Ищем последний tuple в цепочке update'ов.
4) Записываем свой xid в tuple xmax.

Стало лучше – транзакции стали выстраиваться в очередь!

Postgres PROFESSIONAL

HighLoad++ 2017

# Стало лучше, но...




- tuple id меняется и всё очередь постоянно переезжает.
- Получается O($n^2$) операция взятия/отпускаяния лока.
- Heavy weight lock сам по себе не дешёвый. Включается в себя заглядывание в хэш-таблицу в shared memory.

PROFESSIONAL Postgres
HighLoad++ 2017

# Виды lock'ов

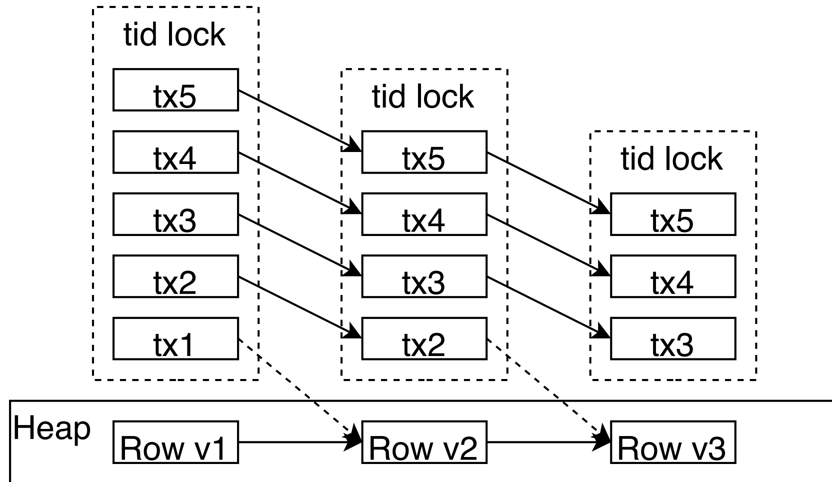
| | spinlock | LWLock | HWLock |
|---|---|---|---|
| Заранее невозможно выделить shared memory под каждый lock | – | – | + |
| Число уровней блокировки | 1 | 2 | 8 |
| Очередь блокировок | – | + | + |
| Deadlock detection | – | – | + |

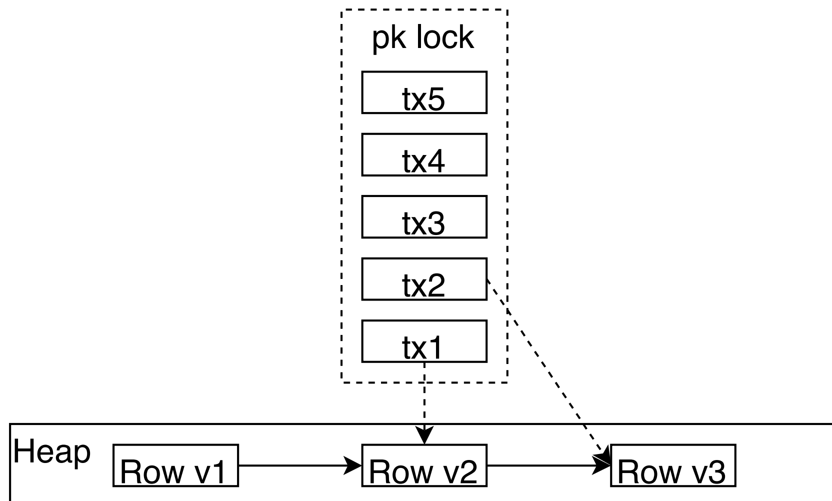
Для tuple lock подходит только heavy-weight lock...

HighLoad++ 2017

PROFESSIONAL Postgres

# Оптимизация: primary key lock




- В отличии от tuple id, значение primary key, как правило, не меняется.
- В итоге очередь на update одна, не переезжает.
- Нужна уметь получать хорошее (и достаточно длинное) хэшированное преставление primary key, чтобы исключить коллизии.

HighLoad++ 2017

# Оптимизации для high-contention write

- Primary key lock during update
- Низкоуровневые оптимизации

  Small improvement to compactify_tuples

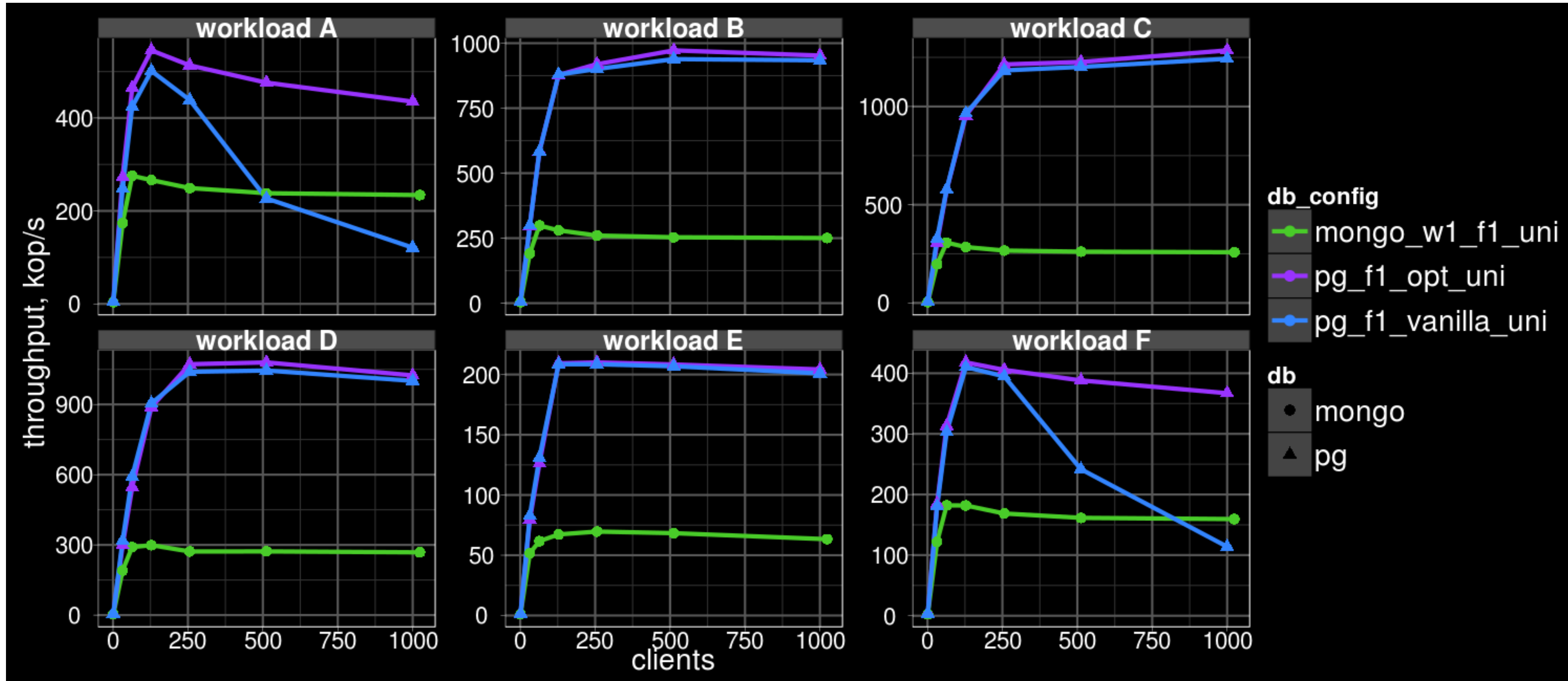
  https://www.postgresql.org/message-id/flat/3c6ff1d3a2ff429ee80d0031e6c69cb7

  Fix performance degradation of contended LWLock on NUMA

  https://www.postgresql.org/message-id/flat/2968c0be065baab8865c4c95de3f435c@postgrespro.ru

Postgres PROFESSIONAL

# Uni with postgres optimized — **GOOD !**

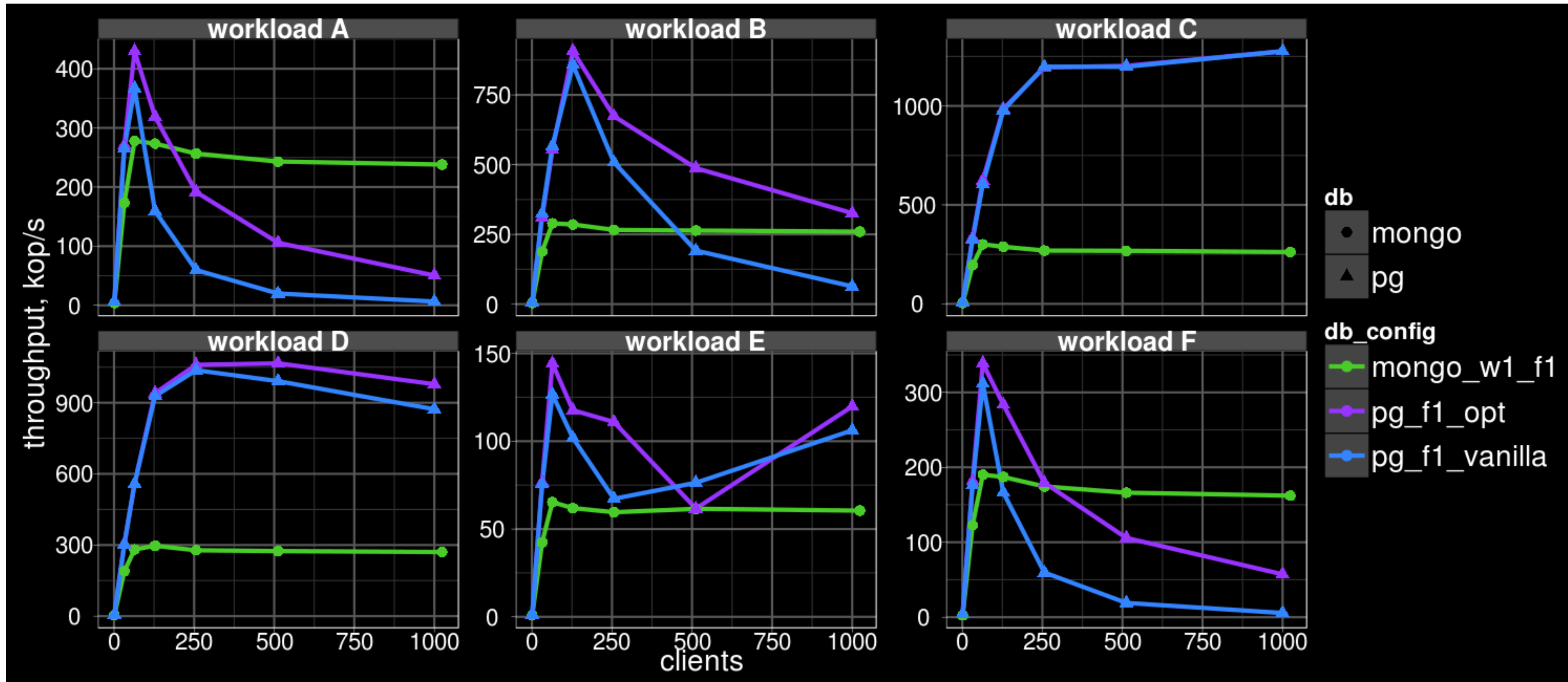



PROFESSIONAL Postgres

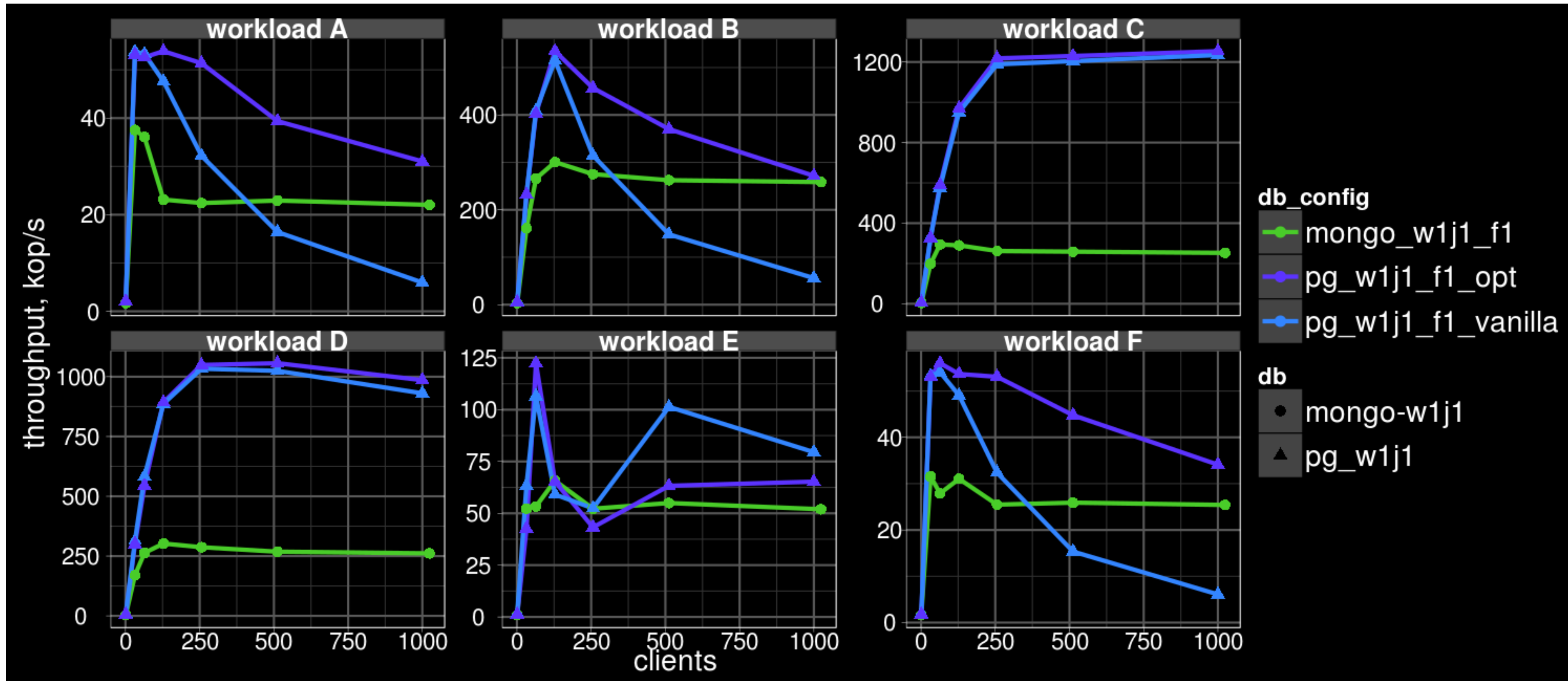
HighLoad++ 2017

PROFESSIONAL Postgres

# Zipf with postgres optimized — **So-So :(**

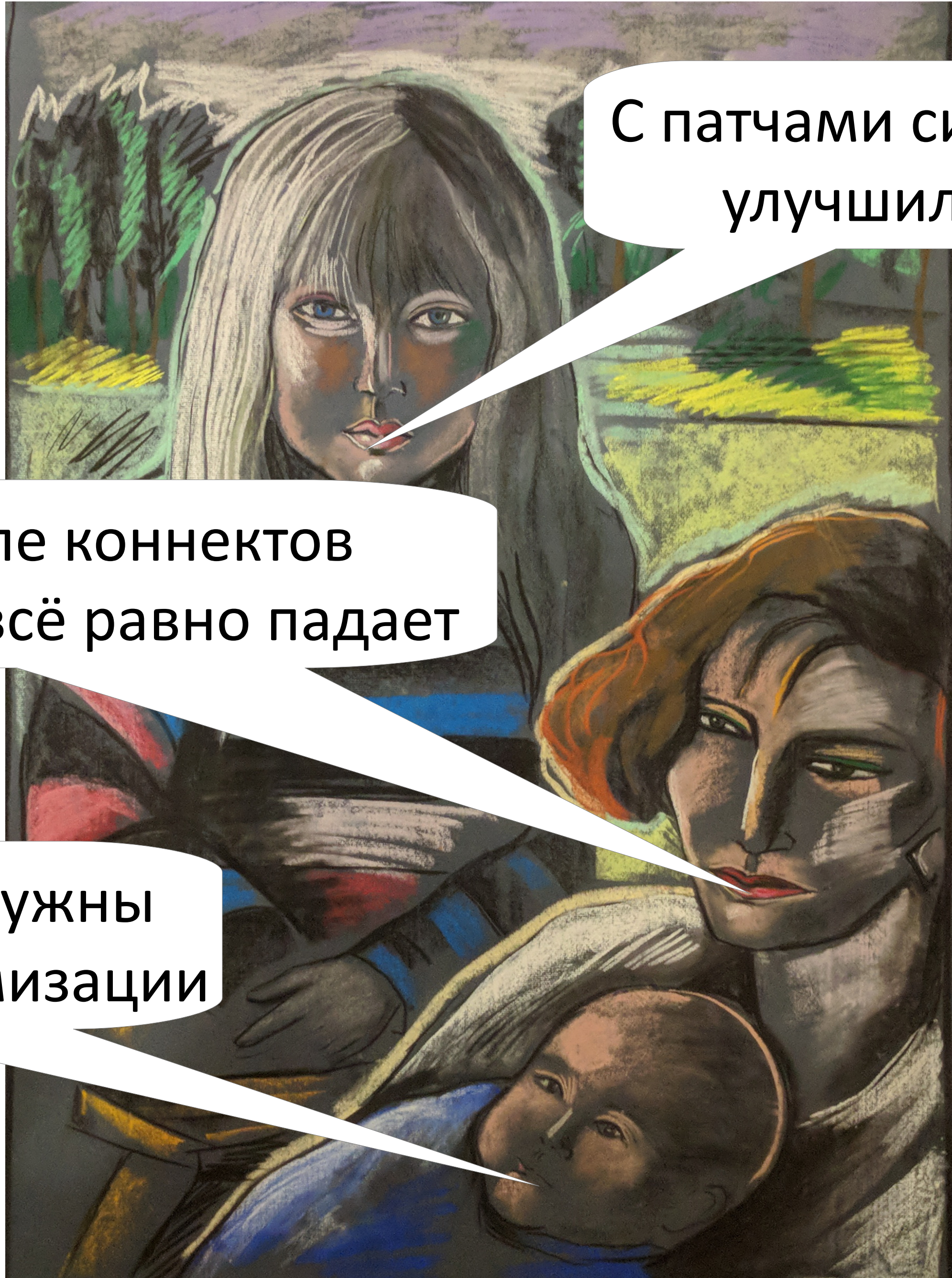


# Persistent case (sync. Commit) — Good :)




HighLoad++ 2017

Postgres PROFESSIONAL



PROFESSIONAL Postgres

# Что делать с high-contention write?

- Для большинства приложений текущей производительности PostgreSQL более чем достаточно.
- По-возможности избегать high-contention write.
- Ждать улучшений :)

# Как избегать high-contention write? (1/4)

```
CREATE TABLE post (
    id bigserial primary key,
    author text,
    title text,
    body text,
    views_count bigint,
    comments_count bigint
);
```

```
CREATE TABLE post_comment (
    id bigserial primary key,
    post_id bigint
        REFERENCES post (id),
    author text,
    body text
);
```

# Как избегать high-contention write? (2/4)

post.views_count

- Строгая consistency, как правило, не нужна
- Накапливаем просмотры в приложении или in-memory БД
- Периодически по cron обновляем значения post.views_count

PROFESSIONAL Postgres

HighLoad++ 2017

# Как избегать high-contention write? (3/4)

```
CREATE TABLE post (
    id bigserial primary key,
    author text,
    title text,
    body text,
    views_count bigint,
    comments_count bigint,
    last_snapshot txid_snapshot
);
```

```
CREATE TABLE post_comment (
    id bigserial primary key,
    post_id bigint
        REFERENCES post (id),
    author text,
    body text,
    insert_xid bigint
);
```

# Как избегать high-contention write? (4/4)

По trigger'у устанавливаем post_comment.insert_xid = txid_current().
По cron'у обновляем счётчик комментариев.

```
UPDATE post
SET comments_count = views_count + (
  SELECT count(*) FROM post_comment pc
  WHERE pc.post_id = post.id AND
    NOT txid_visible_in_snapshot(pc.insert_xid,
                                 post.last_snapshot)
),
last_snapshot = txid_current_snapshot();
```

HighLoad++ 2017

# Выводы

- PostgreSQL показывает хорошие результаты на YCSB benchmark, напрямую конкурируя с NoSQL решениями.
- Если одновременно synchronous_commit = off и используется распределение Зипфа, то возникает проблема с high-contention write.
- Для high-contention write есть ряд патчей, улучшающих ситуацию.
- Можно успешно бороться с high-contention write на уровне приложения, хотя это и костыли.
- Мы всё понимаем, и будем и дальше трудиться над улучшением ситуации.

PROFESSIONAL
Postgres

# СПАСИБО !

HighLoad++ 2017

Postgres PROFESSIONAL