PgBench – Work in Progress

Fabien Coelho

MINES ParisTech, PSL Research University

PostgreSQL Session #9, Paris – November 17, 2017

Pgbench History

PgBench History

2000-

Simple tool based on TPC-B — Tatsuo Ishii 2000

- external... then `contrib/`
- initialization and scale `-i -s 10`
- 2 benchmarks `-t 10000 -c 4`

Tatsuo Ishii by Oleg Bartunov

Visible *and invisible* developments — 2001-

- Initializing
- Scripting
- Running
- Reporting
- Using...
- Debugging
- Refactoring
- Testing

PgBench History

Initializing

`COPY`	initialization	Takahiro Itagaki	2007
`FILLFACTOR`	taux de remplissage	Pavan Deolasee	2007
`UNLOGGED`	tables	Robert Haas	2011
`TABLESPACE`	on tables or index	–	2011
`--foreign-key`	declarations	Jeff Janes	2012
`-I ...`	initialization steps	Masahiko Sawada	2017

PgBench History

Scripting

`-N`	simple update	Tatsuo Ishii	2002
`-f`	script file	Tomoaki Sato	2005
`\set`	basic arithmetic	–	2006
`\sleep`	sleeping	Jan Wieck	2007
`\shell`	shelling...	Michaël Paquier	2009
gaussian	random	Mitsumasa Kondo	2014
exponential	random	Fabien Coelho	2014
expression	integer arithmetic	Robert Haas	2015
double	arithmetic and functions	Fabien Coelho	2016
non-ascii	variable names	–	2017

PgBench History

Running

`-C`	connection	Tatsuo Ishii	2001
`-M`	query mode	Takahiro Itagaki	2008
`-T`	run time	–	2008
`-j`	threading	–	2009
`-R`	throttling	Fabien Coelho	2013
`-L`	latency limit	–	2014
`-f/b ...`	weighted scripts	–	2016

PgBench History

Reporting

`-l`	logging	Neil Conway	2002
`-r`	per statement stats	Florian Pflug	2010
`--sampling-rates`	sample stats	Tomas Vondra	2012
`--aggregate-interval`	aggregated stats	–	2013
`-P`	progress	Fabien Coelho	2013
`-f ...`	per script stats	–	2016

PgBench Capabilities

PgBench Capabilities

`pgbench`

Initialize a database `-i -s 1000 ...`

- create and fill, with scaling
- options: PK, FK, unlogged, fillfactor, tablespace...

Run scripts `-T 1000 -c 32 -j 8 ...`

- psql-like, 3 builtins or custom, weighted, prepared, throttled
- parallelism: threads, clients, re-connections...

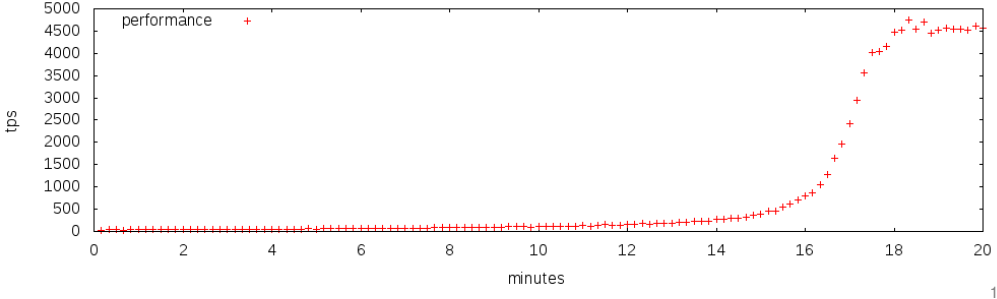
Measure and report performance `-r -l -P 1 ...`

- tps, latency, timeout; per script, per command...
- detailed, sampled or aggregated; stdout or file

Benchmarking Caveats

Beware

- long enough — *warm-up, checkpoint and vacuum*
- several times — *reproducibility*
- representative — *pedal-to-the-metal?*



Benchmarking vs Performance Testing

Benchmarking — *System comparison*

- standard schema and transaction
- maximum load — *pedal to the metal*
- report transaction per second — *tps*
- latency should good enough. . . — *s*

Performance Testing — *Does it work for me?*

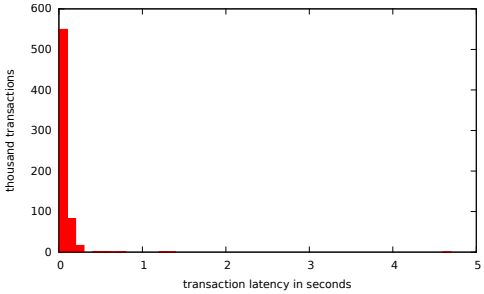
- YOUR schema and transaction
- YOUR load. . . — *throttling*
- load must be processed
- latency must match application constraints

Deceptive Performance

`pgbench -j 4 -c 8`

Version 9.5.5

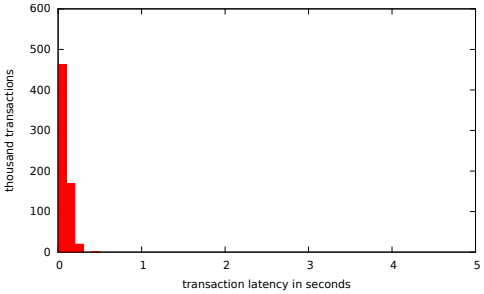
- throughput 329.4 tps
- average latency 24.3 ms



- latency stddev. 79.5 ms

Version 9.6.1

- throughput 326.4 tps
- average latency 24.4 ms

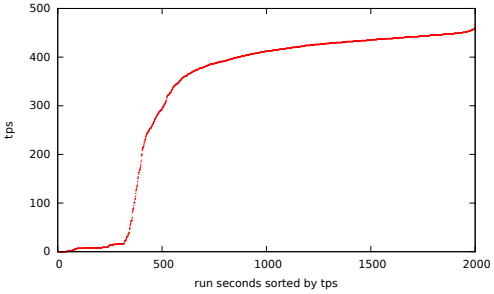


- latency stddev. 20.3 ms

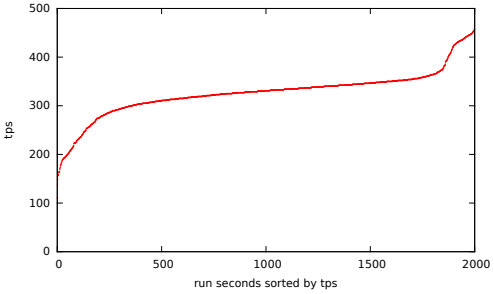
Deceptive Performance

`pgbench -P 1`

Version 9.5.5



Version 9.6.1



What is happening?

Buy Now, Pay Later!

- transaction surges are absorbed — *in-memory + WAL*
- then data are written disk — *checkpoint*

Performance

Tool Overheads

`pgbench -T 10 -P 1 -f script.sql`

Sleep zero — 13.4 Mtps, 75 ns

```
\sleep 0
```

Set a variable — 9.5 Mtps, 105 ns

```
\set i 0
```

Empty command — 97,222 tps, 10.3 μs

```
;
```

Empty SELECT — 51,631 tps, 19.4 μs

```
SELECT;
```

Loading data...

`pgbench -i -I ...`

Impact of schema *on loading time*

steps (d)rop (t)able (g)enerate (v)accum (p)rimary and (f)oreign key

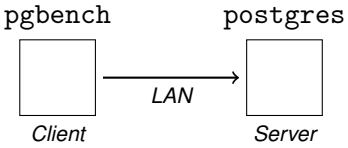
- `pgbench -i -s 100 -I 'dtgv'` 18 s
- `pgbench -i -s 100 -I 'dtgvp'` 29 s
- `pgbench -i -s 100 -I 'dtgvpf'` 32 s
- `pgbench -i -s 100 -I 'dtpgvf'` 39 s
- `pgbench -i -s 100 -I 'dtpfgv'` 103 s

Impact summary

- Primary key 50-100%
- Foreign key 20-300%

Connection Costs

`pgbench -C`



- Client — *8 cores, 16 GB*
- LAN — *1 Gbps*
- Server — *16 cores, 32 GB, HDD*

Initialization and Benchmarks

Postgres 9.6.1

```
pgbench -i -s 100                                          1.5 GB
pgbench -T 2000 -C "host=server sslmode=require"           36.1 tps
pgbench -T 2000 -C "host=server sslmode=disable"           56.4 tps
pgbench -T 2000    "host=server sslmode=disable"           105.4 tps
```

- connection AAA — **8.2 ms**
- SSL negociation — **10.0 ms**
- transfers and transactions — **9.5 ms**

SSL or not?

`pgbench ... "sslmode=..."`

SSL Costs *time & €*

- negotiation and re-negotiation
- cryptographic functions
- certificate ?

Benefits *Snake Oil!*

- Confidentiality
- Integrity
- Authentication

```
pgbench -j 1 -c 1 -D scale=100 -f ro3.sql -T 30 "host=server ..."
```

`sslmode=require` *SSL*

- throughput *709.7 tps*
- latency *1.407 ± 0.132 ms*

`sslmode=disable` *clear*

- throughput *781.6 tps*
- latency *1.277 ± 0.034 ms*

Select Only Index

`pgbench -T 10 -P 1 -S`

With primary key — 17,225 tps

- initialization — `pgbench -i -s 10 -I "dtgvp"`

No primary key — 23 tps

- initialization — `pgbench -i -s 10 -I "dtgv"`

With hash index — 18,289 tps

- initialization — `pgbench -i -s 10 -I "dtgv"`
- plus non unique hash index

```
CREATE INDEX ah ON pgbench_accounts USING HASH(aid);
```

Fill Factor

`pgbench -i -F ...`

Update intensive load *with MVCC*

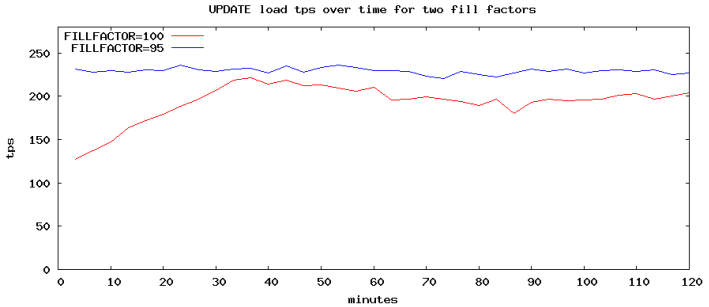
- `UPDATE` = `DELETE` + `INSERT`
- induce about 3 page writes
- *or* keep some free space available

Initialization `pgbench -i -s 100 -F 95`

```
CREATE TABLE pgbench_accounts(...)
  WITH (FILLFACTOR = 95);
...
```

Only UPDATE script

```
\set naccounts 100000 * :scale
\set aid random(1, :naccounts)
\set delta random(-5000, 5000)
UPDATE pgbench_accounts
  SET abalance = abalance + :delta, filler = NOW()::TEXT
  WHERE aid = :aid;
```



Future

PgBench Future

Where to stop?

Needed Capabilities

TPC-B Simple Banking Example *version 2.0, June 1994*

- schema creation and initialization
- simple but non trivial SQL scripting
  - SQL commands (`SELECT INSERT UPDATE COMMIT...`)
  - variables, expressions, (uniform) random numbers, **if**...
- constant performance monitoring and reporting
  stability, steady-state, statistics, errors...
- explicit anti-cheating constraints
  *1.3.2 application **must retrieve** the balance*

Cannot be implemented with PgBench (yet)

Needed Capabilities

Missing features in development

cset/gset	get query results into variables	Fabien Coelho
boolean	comparisons and logical operators	–
if/endif	conditional expression	*refactoring done, waiting on boolean*

Other features in development

pow	another function	Raúl Marín
ppoll	for handling more clients	Doug Rady
stats	initialization step timing	–
perm	pseudo random permutation	...
error	handling...	?

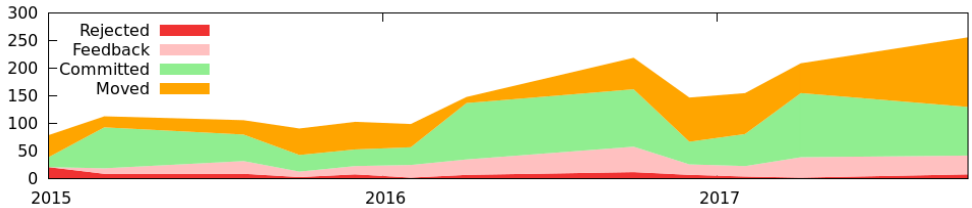
TPC-B *Real* Transaction Profile

```
\set tbid  random(1, :scale)
\set tid  10 * (:tbid - 1) + random(1, 10)
\if :scale = 1 OR random(0, 99) < 85 -- same branch
  \set bid  :tbid
\else -- other branch
  \set bid  1 + (:tbid + random(1, :scale - 1)) % :scale
\endif
\set aid  :bid * 100000 + random(1, 100000)
\set delta  random(-999999, 999999)
BEGIN;
UPDATE pgbench_accounts
  SET abalance = abalance + :delta WHERE aid = :aid
  RETURNING abalance AS balance \gset
UPDATE pgbench_tellers
  SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches
  SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)
  VALUES (:tid, :bid, :aid, :delta, CURRENT_TIMESTAMP);
END;
```

Work in slow Progress

commitfest.postgresql.org

- submit patch — Needs review
- get a review and review others — Needs review | Waiting on Author
- get a decision — Returned with feedback | Ready for Committer | Rejected
- wait for a committer. . . — Moved to next CF
- and maybe get one — Committed | Rejected



Conclusion

Easy to use and improving tool — `pgbench`

- write your custom script
- run it against your data
- for your load

Need test data? — `datafiller`

- directives on SQL declarations
- generators for many types and constraints