Measuring and Reducing Postgres Transaction Latency

(updated version)

Fabien Coelho

MINES ParisTech, PSL Research University

pgDay Paris – March 23, 2017

Talk Outline

1 Introduction
- Subject
- Typical Web Application
- Transaction Performance Definitions
- pgbench
- General Approach

2 Performance Comparisons
- Two Connection Costs
- Latency Pitfalls
- Throughput and Latency Control
- Three Storage Options
- Two Protocol Impacts
- Four Query Combination Tricks
- Reducting Server Distance
- Performance Scalability
- Miscellaneous Settings

3 Conclusion
- Latency and Throughput Wrap-Up
- Lessons Learned
- Contributions to Postgres

Subject

Small OLTP *OnLine Transaction Processing*

- CRUD queries `... WHERE pk=?`
- data fit in shared buffers *small, few GB*
- RW, RO *pgbench builtins*

Focus *and Motivation*

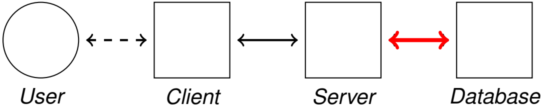
- performance with emphasis on latency *interactive web app*
- experiment & measure *do not assume!*

latency performance : RW ×63, **RO** ×219

Typical Web Application

3-Tier Architecture

Client	user acts on user-agent, sends to
Server	process request, database operations to
Database	stores and retrieves data



Database Operations

- Connection *TCP/IP, SSL & AAA*
- Request-Response **cycles** *transfer, parse, plan, execute, transfer back*

Transaction Performance

Definitions *time & operations*

Throughput operations per time unit *tx/s*
*usual approach, load measured in **tps***

Latency time for one operation *ms/tx*
must fit application requirements

Comments

- correlated *and contradictory*
- max vs enough *and vice-versa*
- sensitive to many settings *net, soft & hard*
- throughput bottleneck & latency additivity *deep voodoo!*

Postgres Performance Swiss Army Knife `pgbench`

Available Features

input	SQL-like scripts with minimal client-side language
options	time to run, prepared, reconnections, ...
parallelism	threads, clients, asynchronous calls
output	statistical performance data

Caveats

- long enough *warm-up, checkpoint and vacuum*
- several times *reproducibility*
- pedal-to-the-metal max speed test *not representative*

Default TPC-B-like Transaction `pgbench -b tcpb-like`

- TPC-B-like banking transaction

```
-- random ids and amount
\set aid random(1, 100000 * :scale)
\set bid random(1, 1 * :scale)
\set tid random(1, 10 * :scale)
\set delta random(-5000, 5000)
-- actual transaction
BEGIN;
UPDATE pgbench_accounts
  SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance
  FROM pgbench_accounts WHERE aid = :aid;
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;
```

Pattern

- 3 updates
- 1 insert
- 1 select

General Approach

Experiment & Measure — *RW or RO*

- one-client runs — *unless otherwise stated*
- independent tests — *one at a time change*
- final wrap up — *cumulative changes*

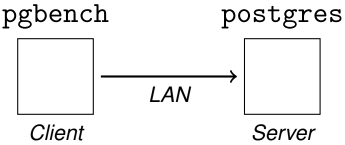
Exploration — *RW or RO*

- two connection costs
- latency pitfalls
- throughput & latency control
- three storage options
- two protocol impacts
- four query combinations
- reducing server distance
- scalability and misc. stuff

Performance Comparisons

Two Connection Costs

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**

Performance Comparisons

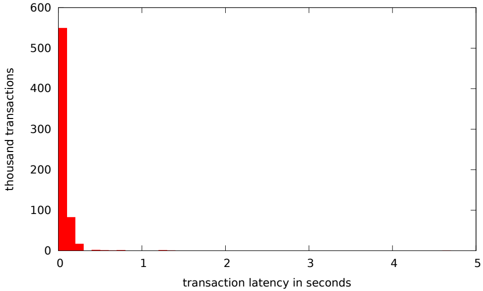
Latency Pitfalls

Latency Comparison – 9.5 vs 9.6

`pgbench -j 4 -c 8`

Version 9.5.5

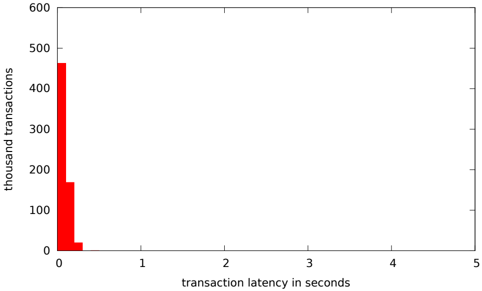
- throughput 329.4 tps
- average latency 24.3 ms



- latency std. dev. 79.5 ms

Version 9.6.1

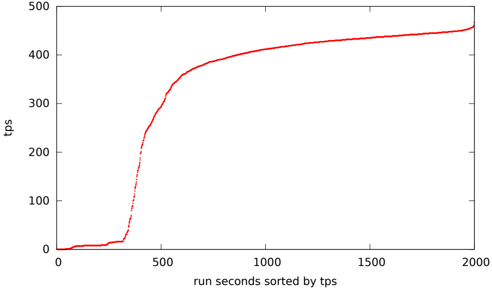
- throughput 326.4 tps
- average latency 24.4 ms



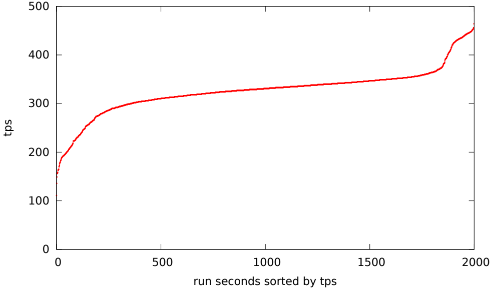
- latency std. dev. 20.3 ms

Latency Comparison – 9.5 vs 9.6 *Instant TPS*

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*

Latency Comparison – 9.5 vs 9.6 *Checkpointing*

Postgres 9.5 Checkpoint

- data writes spread over some time *random I/O*
- OS choose when to actually write *30s delay on Linux*
- until `fsync` is called. . . ***I/O storm – on low-end HDD***

Postgres 9.6 Checkpoint

- **sorted** data writes spread over some time *sequential I/O*
- **flush** instructions sent regularly (256 kB) *checkpoint_flush_after*
- when `fsync` is called *ok!*

Performance Comparisons

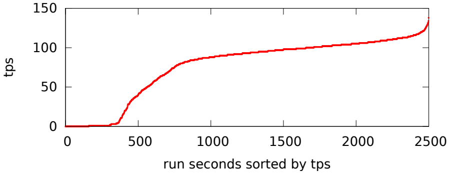
Throughput and Latency Control

Rate (tps) and Limit (ms)

`pgbench -R 100 -L 100 -N`

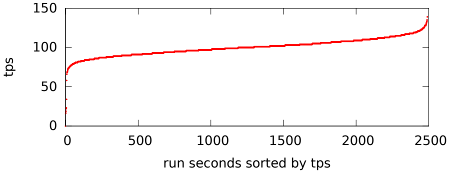
Pg 9.5 *basic checkpoint*

- slow & skipped *24.0%*
- latency *15.6 ± 158.3 ms*



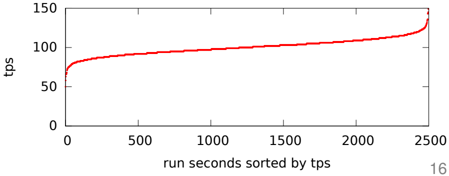
Pg 9.6 *sorted checkpoint*

- slow & skipped *2.7%*
- latency *3.6 ± 24.6 ms*



Pg 9.6 *sorted & flushed checkpoint*

- slow & skipped *0.5%*
- latency *2.6 ± 13.8 ms*



Performance Comparisons

Three Storage Options

FILLFACTOR Storage Parameter

```
CREATE TABLE pgbench_accounts(...) WITH (FILLFACTOR = 100);
```

FILLFACTOR Usage

- MVCC: UPDATE = DELETE + INSERT *up to 3 pages changes*
- some free space available in page *1 inside page change*
- **but** more pages/costs for other operations *trade-off*

FILLFACTOR = 100

- throughput *406.9 tps*
- latency *19.7 ± 12.3 ms*

FILLFACTOR = 95

- throughput *416.8 tps*
- latency *19.2 ± 8.3 ms*

Hardware *HDD vs SSD*

Hard Disk Drive ●

- mechanics
- fast sequential I/O
- **slow** random I/O

vs

Solid State Disk ●

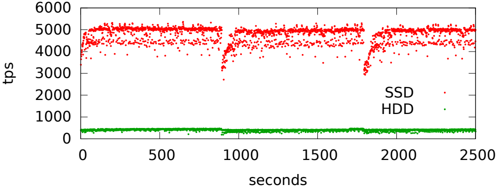
- electronics
- fast sequential I/O
- **fast** random I/O

```
pgbench -j 4 -c 8 -T 2500 -M prepared ...
```

Postgres 9.6

HDD	406.9 tps	*19.7 ± 12.3 ms*
SSD	4,764.9 tps	*1.7 ± 2.4 ms*

checkpoint full page write effect



UNLOGGED TABLE *Can you loose your data?*

```
CREATE UNLOGGED TABLE pgbench_accounts(...);
```

Standard	*ACID*
■ throughput	*406.9 tps*
■ latency	*19.7 ± 12.3 ms*

UNLOGGED	*good luck!*
■ throughput	*5,310.7 tps*
■ latency	*1.5 ± 0.3 ms*

NO!

Performance Comparisons

Two Protocol Impacts

Read-Only In-Cache Test *ro3.sql*

```
\set aid random(1, 100000 * :scale)
\set tid random(1, 10 * :scale)
\set bid random(1, :scale)
BEGIN;
SELECT abalance FROM pgbench_accounts WHERE aid=:aid;
SELECT tbalance FROM pgbench_tellers WHERE tid=:tid;
SELECT bbalance FROM pgbench_branches WHERE bid=:bid;
COMMIT;
```

Operations	*Queries on 3 tables*
1 transfers	*network protocol*
2 parse query	*syntax analysis*
3 plan query	*optimization*
4 execute query	*cheap if in cache*

Protocol *SSL or not*

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*

Protocol *Simple vs Prepared*

```
-- prepare once in session
PREPARE Abal(INT) AS
  SELECT abalance
  FROM pgbench_accounts
  WHERE aid=$1;
-- execute multiple times...
EXECUTE Abal(1);
EXECUTE Abal(5432);
EXECUTE Abal(18);
```

Prepare

- temporary one-cmd function
- factor out *parse* cost
- keep *plan* and *execute*
- `pgbench -M prepared ...`

`ro3.sql` *simple*

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

`ro3.sql` *prepared*

- throughput *860.0 tps*
- latency *1.161 ± 0.082 ms*

Performance Comparisons

Four Query Combination Tricks

Query Combination

UPDATE & SELECT

```
-- update table
UPDATE pgbench_accounts
  SET abalance = abalance + :delta
  WHERE aid = :aid;

-- get updated data
SELECT abalance
  FROM pgbench_accounts
  WHERE aid = :aid;
```

```
-- combined
UPDATE pgbench_accounts
  SET abalance = abalance + :delta
  WHERE aid = :aid
  RETURNING abalance;
```

UPDATE RETURNING Option

- return updated rows
- one parse, plan, execute

Standard

- throughput *406.9 tps*
- latency *19.7 ± 12.3 ms*

Combined Update

- throughput *408.2 tps*
- latency *19.6 ± 8.7 ms*

Client-combined SQL Queries

```
-- "ro3c.sql" pgbench script
\set aid random(1, 100000 * :scale)
\set tid random(1, 10 * :scale)
\set bid random(1, :scale)
BEGIN \;
SELECT abalance FROM
 pgbench_accounts WHERE aid=:aid \;
SELECT tbalance FROM
 pgbench_tellers WHERE tid=:tid \;
SELECT bbalance FROM
 pgbench_branches WHERE bid=:bid \;
COMMIT;
```

Combine *with \;*

- embedded semi-colon ;
- request with multiple queries
- response with list of results
- avoid request-response loop

`ro3.sql` *standard*

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

`ro3c.sql` *combined*

- throughput *1,311.5 tps*
- latency *0.748 ± 0.132 ms*

Server-Side SQL queries

```
 CREATE TYPE Balances
   AS (abal INT, tbal INT, bbal INT);

CREATE FUNCTION getBalSQL(INT, INT, INT)
  RETURNS Balances AS $$
  SELECT
    (SELECT abalance
     FROM pgbench_accounts WHERE aid=$1),
    (SELECT tbalance
     FROM pgbench_tellers WHERE tid=$2),
    (SELECT bbalance
     FROM pgbench_branches WHERE bid=$3)
$$ LANGUAGE SQL;
```

```
 -- "ro3sf.sql" pgbench script
\set aid random(1, 100000 * :scale)
\set tid random(1, 10 * :scale)
\set bid random(1, :scale)
SELECT getBalSQL(:aid, :tid, :bid);
```

`ro3.sql` *standard*

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

`ro3sf.sql` *SQL call*

- throughput *1,395.4 tps*
- latency *0.712 ± 0.075 ms*

Server-Side PL/pgSQL queries

```
  CREATE FUNCTION
      getBalPL(a INT, t INT, b INT)
    RETURNS Balances AS $$
    DECLARE
      abal INT; tbal INT; bbal INT;
    BEGIN
      SELECT abalance INTO abal
        FROM pgbench_accounts WHERE aid=a;
      SELECT tbalance INTO tbal
        FROM pgbench_tellers WHERE tid=t;
      SELECT bbalance INTO bbal
        FROM pgbench_branches WHERE bid=b;
      RETURN (abal, tbal, bbal)::Balances;
    END;
$$ LANGUAGE PLpgSQL;
```

```
 -- ”ro3pf.sql” pgbench script
\set aid random(1, 100000 * :scale)
\set tid random(1, 10 * :scale)
\set bid random(1, :scale)
SELECT getBalPL(:aid, :tid, :bid);
```

PL/pgSQL caches plans!

ro3.sql *standard*

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

ro3pf.sql *PL/pgSQL call*

- throughput *2,485.5 tps*
- latency *0.400 ± 0.055 ms*

Performance Comparisons

Reducting Server Distance

Client-Server Distance

Interconnection

LAN	Local Area Network	*Ethernet*
LO	loopback interface	`localhost`
IPC	Inter-Process Communication	*Unix domain socket*

TPC-B-Like *on SSD*

LAN	*403.8 tps*	*2.4 ms*
LO	*1,133.3 tps*	*0.9 ms*
IPC	*1,243.1 tps*	*0.8 ms*

Read-Only 3

LAN	*709.7 tps*	*1.4 ms*
LO	*2,515.3 tps*	*0.4 ms*
IPC	*3,607.6 tps*	*0.3 ms*

Performance Comparisons

Performance Scalability

# Clients Scalability

Base

Best Throughput ●

37,639 tps 4.103 ms *156/4*

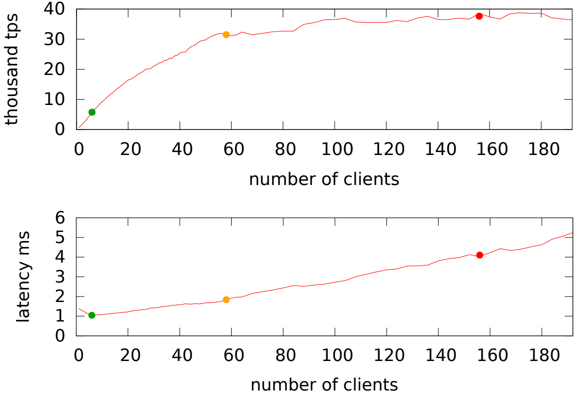
Best Latency ●

5,748 tps **1.042 ms** *6/1*

Compromise ●

31,494 tps 1.837 ms *58/4*

Read-Only 3 – remote SSL simple queries



# Clients Scalability *Best*

Read-Only 3 – remote noSSL prepared PL call

Best Throughput ●

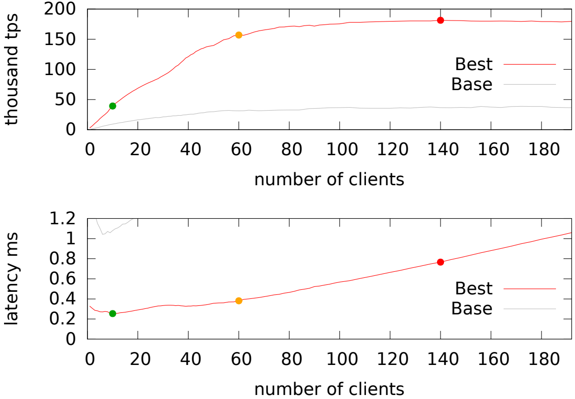
181,503 tps 0.766 ms *140/4*

Best Latency ●

39,232 tps **0.254 ms** *10/2*

Compromise ●

156,945 tps 0.381 ms *60/4*



Performance Comparisons

Miscellaneous Settings

Miscellaneous Settings *App & Postgres*

Application *framework?*

connection	persistence
cache	Memcached Redis

Postgres configuration *change defaults*

disk	*block_size random_page_cost*
memory	*shared_buffers effective_cache_size huge_pages*
checkpoint	*_timeout _completion_target _flush_after*
wal	*max_wal_size*

Miscellaneous Settings *OS & Hardware*

OS *tweak and choose*

FS **XFS** ext4 ~~Btrfs~~ ~~ZFS~~, `mount` options

IO io scheduler, queue length, write delay, dirty bytes. . .

others NUMA, . . .

Hardware *expensive is (probably) better*

diskS tables wal logs, HDD-with-cache, SSD

tweaking read ahead, write flush

RAID with large caches, BBU

Conclusion

Wrap-up `pgbench -j 1 -c 1 ...`

	TPC-B-like		*Read-Only 3*	
	tps	*ms*	*tps*	*ms*
HDD `-C` SSL	36.1	27.7	52.7	18.96
HDD `-C` noSSL	56.4	17.7	110.1	9.08
HDD SSL	105.4	9.5	709.7	1.41
SSD SSL	403.8	2.47	695.1	1.44
SSD noSSL	465.4	2.15	820.1	1.22
. . . + prepared	548.1	1.82	974.0	1.02
– returning	529.4	1.89	–	–
. . . + prepared	681.2	1.47	–	–
– combined	857.8	1.15	1,536.4	0.64
– SQL func	940.3	1.06	1,818.1	0.55
. . . + prepared	957.9	1.04	2,144.7	0.46
– PL func	1,279.4	0.78	2,778.0	0.36
. . . + prepared	1,323.2	0.75	3,040.4	0.33
localhost	1,907.6	0.52	10,006.8	0.10
socket	2,273.1	0.44	11,545.5	0.09

- connection
- HDD to SSD
- SSL to none
- simple to prepared
- combinations. . .
- remote to local

× **63 to** × **219**

- *and scaling effects*

Lessons

Things to Bring Home *in-memory OLTP load*

NoTPS	not only TPS latency-throughput compromise	*latency matters!*
Performance	experiment and measure `pgbench` is improving...	*do not assume!*
Postgres	version sorted and flushed checkpoints	*9.6!*
High	costs	*network, parse & plan*
RW load	ACID	*SSD $\gg$ HDD*
RO load	pg as a cache manager	*SSD $=$ HDD*

Contributions *provided or provoked*

About Core *& Andres Freund*

- sorted checkpoints
- flushed checkpoints

About `pgbench` *& Robert Haas*

- expressions `\set ...`
- mixed and weighted scripts and builtins `-b/-f ...@...`
- better statistics *stddev, per script...*
- improved usability `-c/-j -P...`
- rate and limit load `-R -L`
- debug...

Measuring and Reducing Postgres Transaction Latency

(updated version)

Fabien Coelho

MINES ParisTech, PSL Research University

pgDay Paris – March 23, 2017