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

<table>
<thead>
<tr>
<th>Small OLTP</th>
<th>OnLine Transaction Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>- CRUD queries</td>
<td>... WHERE pk=?</td>
</tr>
<tr>
<td>- data fit in shared buffers</td>
<td>small, few GB</td>
</tr>
<tr>
<td>- RW, RO</td>
<td>pgbench builtins</td>
</tr>
</tbody>
</table>

Focus and Motivation

- performance with emphasis on latency
- experiment & measure

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
- Request-Response cycles

TCP/IP, SSL & AAA
transfer, parse, plan, execute, transfer back
## Transaction Performance

### Definitions

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>operations per time unit&lt;br&gt;&lt;i&gt;usual approach, load measured in tps&lt;/i&gt;</td>
</tr>
<tr>
<td>Latency</td>
<td>time for one operation&lt;br&gt;&lt;i&gt;must fit application requirements&lt;/i&gt;</td>
</tr>
</tbody>
</table>

### Comments

- correlated
- max vs enough
- sensitive to many settings
- throughput bottleneck & latency additivity

<i>and contradictory</i><br><i>and vice-versa</i><br><i>net, soft & hard</i><br><i>deep voodoo!</i>
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
- several times
- pedal-to-the-metal max speed test

*warm-up, checkpoint and vacuum reproducibility not representative*
Default TPC-B-like Transaction

TPC-B-like banking transaction

```sql
-- 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**

- one-client runs
- independent tests
- final wrap up

**Exploration**

- two connection costs
- latency pitfalls
- throughput & latency control
- three storage options

- two protocol impacts
- four query combinations
- reducing server distance
- scalability and misc. stuff

**RW or RO**

- unless otherwise stated
- one at a time change
- cumulative changes

**Conclusion**

- Wrap-Up
- Lessons
- Contributions
Performance Comparisons

Two Connection Costs
Connection Costs

Postgres Latency

F. Coelho

Introduction
Subject
Application
Definitions
pgbench
Approach
Performance
Connection
Latency
Rate & Limit
Storage
Protocol
Combinations
Distance
Scalability
Miscellaneous

Conclusion
Wrap-Up
Lessons
Contributions

pgbench -C

Client
LAN
Server

8 cores, 16 GB
1 Gbps
16 cores, 32 GB, HDD

Postgres 9.6.1

Initialization and Benchmarks

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
- SSL negotiation
- transfers and transactions

8.2 ms
10.0 ms
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
What is happening?

- transaction surges are absorbed
- then data are written disk

Buy Now, Pay Later!

*in-memory + WAL checkpoint*
Latency Comparison – 9.5 vs 9.6

**Checkpointing**

**Postgres 9.5 Checkpoint**
- data writes spread over some time
- OS choose when to actually write
- until `fsync` is called…

**Postgres 9.6 Checkpoint**
- **sorted** data writes spread over some time
- **flush** instructions sent regularly (256 kB)
- when `fsync` is called

---

**Latency Comparison – 9.5 vs 9.6**

**Postgres 9.5 Checkpoint**
- random I/O
- 30s delay on Linux

**Postgres 9.6 Checkpoint**
- sequential I/O
  - `checkpoint_flush_after`
  - ok!
Performance Comparisons

Throughput and Latency Control
Rate (tps) and Limit (ms)

**Pg 9.5**  
*basic checkpoint*
- slow & skipped: 24.0%
- latency: $15.6 \pm 158.3$ ms

**Pg 9.6**  
*sorted checkpoint*
- slow & skipped: 2.7%
- latency: $3.6 \pm 24.6$ ms

**Pg 9.6**  
*sorted & flushed checkpoint*
- slow & skipped: 0.5%
- latency: $2.6 \pm 13.8$ ms
Performance Comparisons

Three Storage Options
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**

<table>
<thead>
<tr>
<th>Hard Disk Drive</th>
<th>Solid State Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>mechanics</td>
<td>electronics</td>
</tr>
<tr>
<td>fast sequential I/O</td>
<td>fast sequential I/O</td>
</tr>
<tr>
<td><strong>slow</strong> random I/O</td>
<td><strong>fast</strong> random I/O</td>
</tr>
</tbody>
</table>

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

**Postgres 9.6**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>406.9 tps</td>
<td>19.7 ± 12.3 ms</td>
</tr>
<tr>
<td>SSD</td>
<td>4,764.9 tps</td>
<td>1.7 ± 2.4 ms</td>
</tr>
</tbody>
</table>

*checkpoint full page write effect*
CREATE UNLOGGED TABLE pgbench_accounts(...);

<table>
<thead>
<tr>
<th></th>
<th>Standard</th>
<th>ACID</th>
<th>UNLOGGED</th>
<th>good luck!</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>406.9 tps</td>
<td></td>
<td>5,310.7 tps</td>
<td></td>
</tr>
<tr>
<td>latency</td>
<td>19.7 ± 12.3 ms</td>
<td></td>
<td>1.5 ± 0.3 ms</td>
<td></td>
</tr>
</tbody>
</table>

UNLOGGED TABLE

Can you lose your data?

NO!
Performance Comparisons

Two Protocol Impacts
Read-Only In-Cache Test

\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

1. transfers
2. parse query
3. plan query
4. execute query

Queries on 3 tables

network protocol
syntax analysis
optimization
cheap if in cache
## Protocol

### SSL Costs

<table>
<thead>
<tr>
<th>Negotiation and re-negotiation</th>
<th>Cryptographic functions</th>
<th>Certificate?</th>
</tr>
</thead>
</table>

### Benefits

- Confidentiality
- Integrity
- Authentication

---

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

### SSL vs Clear

<table>
<thead>
<tr>
<th>SSL Costs</th>
<th>Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>709.7 tps</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>1.407 ± 0.132 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSL Costs</th>
<th>Clear</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>781.6 tps</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>1.277 ± 0.034 ms</td>
</tr>
</tbody>
</table>
Protocol

--- 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` ...

<table>
<thead>
<tr>
<th>ro3.sql</th>
<th>simple</th>
<th>prepared</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>709.7 tps</td>
<td>860.0 tps</td>
</tr>
<tr>
<td>latency</td>
<td>1.407 ± 0.132 ms</td>
<td>1.161 ± 0.082 ms</td>
</tr>
</tbody>
</table>
Performance Comparisons

Four Query Combination Tricks
### Query Combination

--- update table

```sql
UPDATE pgbench_accounts
SET abalance = abalance + :delta
WHERE aid = :aid;
```

--- get updated data

```sql
SELECT abalance
FROM pgbench_accounts
WHERE aid = :aid;
```

--- combined

```sql
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

```sql
-- "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

<table>
<thead>
<tr>
<th>ro3.sql</th>
<th>standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>709.7 tps</td>
</tr>
<tr>
<td>latency</td>
<td>1.407 ± 0.132 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ro3c.sql</th>
<th>combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>1,311.5 tps</td>
</tr>
<tr>
<td>latency</td>
<td>0.748 ± 0.132 ms</td>
</tr>
</tbody>
</table>
Server-Side SQL queries

```sql
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);
```

### Performance

<table>
<thead>
<tr>
<th>ro3.sql</th>
<th>standard</th>
<th>ro3sf.sql</th>
<th>SQL call</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>709.7 tps</td>
<td>throughput</td>
<td>1,395.4 tps</td>
</tr>
<tr>
<td>latency</td>
<td>1.407 ± 0.132 ms</td>
<td>latency</td>
<td>0.712 ± 0.075 ms</td>
</tr>
</tbody>
</table>
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!

<table>
<thead>
<tr>
<th>ro3.sql</th>
<th>standard</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>709.7 tps</td>
</tr>
<tr>
<td>latency</td>
<td>1.407 ± 0.132 ms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ro3pf.sql</th>
<th>PL/pgSQL call</th>
</tr>
</thead>
<tbody>
<tr>
<td>throughput</td>
<td>2,485.5 tps</td>
</tr>
<tr>
<td>latency</td>
<td>0.400 ± 0.055 ms</td>
</tr>
</tbody>
</table>
Performance Comparisons

Reducing Server Distance
Client-Server Distance

Interconnection

<table>
<thead>
<tr>
<th>Interconnection</th>
<th>TPC-B-Like</th>
<th>on SSD</th>
<th>Read-Only 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAN</td>
<td>403.8 tps</td>
<td>2.4 ms</td>
<td>LAN</td>
</tr>
<tr>
<td></td>
<td>1,133.3 tps</td>
<td>0.9 ms</td>
<td>709.7 tps</td>
</tr>
<tr>
<td>LO</td>
<td></td>
<td></td>
<td>LO</td>
</tr>
<tr>
<td></td>
<td>1,243.1 tps</td>
<td>0.8 ms</td>
<td>2,515.3 tps</td>
</tr>
<tr>
<td>IPC</td>
<td></td>
<td></td>
<td>IPC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3,607.6 tps</td>
</tr>
</tbody>
</table>

LAN: Local Area Network
LO: loopback interface
IPC: Inter-Process Communication
Ethernet
localhost
Unix domain socket
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**
# Clients Scalability

**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

---

Read-Only 3 – remote noSSL prepared PL call
Performance Comparisons

Miscellaneous Settings
## Miscellaneous Settings

### Application

<table>
<thead>
<tr>
<th>connection</th>
<th>persistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache</td>
<td>Memcached Redis</td>
</tr>
</tbody>
</table>

### Postgres configuration

<table>
<thead>
<tr>
<th>disk</th>
<th>block_size random_page_cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>memory</td>
<td>shared_buffers effective_cache_size huge_pages</td>
</tr>
<tr>
<td>checkpoint</td>
<td>_timeout _completion_target _flush_after</td>
</tr>
<tr>
<td>wal</td>
<td>max_wal_size</td>
</tr>
</tbody>
</table>

---

**App & Postgres**

### framework?

- cache Memcached Redis

### 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
### OS & Hardware

#### OS
- **FS**: XFS, ext4, Btrfs, ZFS, mount options
- **IO**: io scheduler, queue length, write delay, dirty bytes...
- **others**: NUMA, ...

#### Hardware
- **diskS**: tables wal logs, HDD-with-cache, SSD
- **tweaking**: read ahead, write flush
- **RAID**: with large caches, BBU
Conclusion
### Postgres Latency

F. Coelho

#### Introduction

- Subject
- Application
- Definitions
- pgbench
- Approach

#### Performance

- Connection
- Latency
- Rate & Limit
- Storage
- Protocol
- Combinations
- Distance
- Scalability
- Miscellaneous

#### Conclusion

- Wrap-Up
- Lessons
- Contributions

---

#### TPC-B-like

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

---

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

\[ \times 63 \text{ to } \times 219 \]

- *and scaling effects*
## Lessons

### Things to Bring Home

<table>
<thead>
<tr>
<th>NoTPS</th>
<th>not only TPS</th>
<th>latency-throughput compromise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>experiment and measure</td>
<td>do not assume!</td>
</tr>
<tr>
<td>Postgres</td>
<td>version</td>
<td>9.6!</td>
</tr>
<tr>
<td>High</td>
<td>costs</td>
<td>network, parse &amp; plan</td>
</tr>
<tr>
<td>RW load</td>
<td>ACID</td>
<td>SSD ≫ HDD</td>
</tr>
<tr>
<td>RO load</td>
<td>pg as a cache manager</td>
<td>SSD = HDD</td>
</tr>
</tbody>
</table>
### Contributions

**provided or provoked**

#### About Core

- sorted checkpoints
- flushed checkpoints

#### About pgbench

- expressions
- mixed and weighted scripts and builtins
- better statistics
- improved usability
- rate and limit load
- debug...

\[ \text{set ...} \]

[set: `\set`]

\[-b/-f \ldots\]@...\[

[set: `-b/-f @...`

\[\text{stddev, per script...}\]

[set: `stddev, per script...`]

\[-c/-j -P...\]

[set: `-c/-j -P...`]

\[-R -L\]

[set: `-R -L`]

---

---
Measuring and Reducing Postgres Transaction Latency

(updated version)

Fabien Coelho

MINES ParisTech, PSL Research University

pgDay Paris – March 23, 2017