Solving PostgreSQL wicked problems

Alexander Korotkov

Oriole DB Inc.

2021
PostgreSQL has two sides
The bright side of PostgreSQL
PostgreSQL – one of the most popular DBMS’es

<table>
<thead>
<tr>
<th>Rank</th>
<th>Jan 2021</th>
<th>Dec 2020</th>
<th>Jan 2020</th>
<th>DBMS</th>
<th>Score</th>
<th>Jan 2021</th>
<th>Dec 2020</th>
<th>Jan 2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Oracle</td>
<td>1322.93</td>
<td>-2.66</td>
<td>-23.75</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>MySQL</td>
<td>1252.06</td>
<td>-3.40</td>
<td>-22.60</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>Microsoft SQL Server</td>
<td>1031.23</td>
<td>-6.85</td>
<td>-67.31</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>PostgreSQL</td>
<td>552.23</td>
<td>+4.65</td>
<td>+45.03</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>MongoDB</td>
<td>457.22</td>
<td>-0.51</td>
<td>+30.26</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>IBM Db2</td>
<td>157.17</td>
<td>-3.26</td>
<td>-11.53</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>8</td>
<td></td>
<td>Redis</td>
<td>155.01</td>
<td>+1.38</td>
<td>+6.26</td>
<td></td>
</tr>
</tbody>
</table>

1According to db-engines.com
PostgreSQL – strong trend

DB-Engines Ranking of PostgreSQL

© August 2021, DB-Engines.com

https://db-engines.com/en/ranking_trend/system/PostgreSQL
PostgreSQL – most loved RDBMS

<table>
<thead>
<tr>
<th>Database</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redis</td>
<td>66.5%</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>63.9%</td>
</tr>
<tr>
<td>Elasticsearch</td>
<td>58.7%</td>
</tr>
<tr>
<td>MongoDB</td>
<td>56.0%</td>
</tr>
<tr>
<td>Firebase</td>
<td>54.9%</td>
</tr>
<tr>
<td>MariaDB</td>
<td>51.3%</td>
</tr>
</tbody>
</table>

---

3 According to Stackoverflow 2020 survey
The dark side of PostgreSQL
10 Things I Hate About PostgreSQL

Over the last few years, the software development community’s love affair with the popular open-source relational database has reached a bit of a fever pitch. This Hacker News thread covering a piece titled “PostgreSQL is the worlds’ best database”, busting at the seams with fawning sycophants lavishing unconditional praise, is a perfect example of this phenomenon.

https://medium.com/@rbranson/10-things-i-hate-about-postgresql-20dbab8c2791
<table>
<thead>
<tr>
<th>Problem name</th>
<th>Known for</th>
<th>Work started</th>
<th>Resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Wraparound</td>
<td>20 years</td>
<td>15 years ago</td>
<td>Still WIP</td>
</tr>
<tr>
<td>2. Failover Will Probably Lose Data</td>
<td>20 years</td>
<td>16 years ago</td>
<td>Still WIP</td>
</tr>
<tr>
<td>3. Inefficient Replication That Spreads Corruption</td>
<td>10 years</td>
<td>8 years ago</td>
<td>Still WIP</td>
</tr>
<tr>
<td>4. MVCC Garbage Frequently Painful</td>
<td>20 years</td>
<td>19 years ago</td>
<td>Abandoned</td>
</tr>
<tr>
<td>5. Process-Per-Connection = Pain at Scale</td>
<td>20 years</td>
<td>3 years ago</td>
<td>Abandoned</td>
</tr>
<tr>
<td>6. Primary Key Index is a Space Hog</td>
<td>13 years</td>
<td>—</td>
<td>Not started</td>
</tr>
<tr>
<td>7. Major Version Upgrades Can Require Downtime</td>
<td>21 years</td>
<td>16 years ago</td>
<td>Still WIP</td>
</tr>
<tr>
<td>8. Somewhat Cumbersome Replication Setup</td>
<td>10 years</td>
<td>9 years ago</td>
<td>Still WIP</td>
</tr>
<tr>
<td>9. Ridiculous No-Planner-Hints Dogma</td>
<td>20 years</td>
<td>11 years ago</td>
<td>Extension</td>
</tr>
<tr>
<td>10. No Block Compression</td>
<td>12 years</td>
<td>11 years ago</td>
<td>Still WIP</td>
</tr>
</tbody>
</table>

* Scalability on modern hardware
PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market.
PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market. As a result, PostgreSQL has had a **strong upwards trend** for many years.
PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market. As a result, PostgreSQL has had a strong upwards trend for many years. At the same time, the PostgreSQL community appears to be dysfunctional in solving design issues, attracting severe criticism. Nevertheless, critics not yet break the upwards trend.
PostgreSQL community have proven to be brilliant on solving non-design issues, providing fantastic product to the market. As a result, PostgreSQL has had a **strong upwards trend** for many years.

At the same time, the PostgreSQL community appears to be dysfunctional in solving design issues, attracting severe criticism. Nevertheless, critics **not yet** break the upwards trend.

It appears to be a **unique moment** for PostgreSQL redesign!
How could we solve the PostgreSQL wicked problems?
Traditional buffer management

<table>
<thead>
<tr>
<th>Disk</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>1'</td>
<td>1</td>
</tr>
<tr>
<td>2'</td>
<td>2</td>
</tr>
<tr>
<td>3'</td>
<td>3</td>
</tr>
<tr>
<td>5'</td>
<td>4</td>
</tr>
<tr>
<td>6'</td>
<td>5</td>
</tr>
<tr>
<td>Memory mapping: 1 2 3 5 6 4 7</td>
<td></td>
</tr>
</tbody>
</table>

- Each page access requires lookup into buffer mapping data structure.
- Each B-tree key lookup takes multiple buffer mapping lookups.
- Accessing cached data doesn’t scale on modern hardware.
Each page access requires lookup into buffer mapping data structure.
Each B-tree key lookup takes multiple buffer mapping lookups.
Each page access requires lookup into buffer mapping data structure.
Each B-tree key lookup takes multiple buffer mapping lookups.
Accessing cached data doesn’t scale on modern hardware.
Solution: Dual pointers

- In-memory page refers either in-memory or on-disk page.
Solution: Dual pointers

- In-memory page refers either in-memory or on-disk page.
- Accessing cached data without buffer mapping lookups.
Solution: Dual pointers

- In-memory page refers either in-memory or on-disk page.
- Accessing cached data without buffer mapping lookups.
- Good scalability!
PostgreSQL MVCC = bloat + write-amplification

- New and old row versions shares the same heap.
PostgreSQL MVCC = bloat + write-amplification

- New and old row versions shares the same heap.
- Non-HOT updates cause index bloat.
Solution: undo log for both pages and rows

- Old row versions form chains in undo log.
Solution: undo log for both pages and rows

- Old row versions form chains in undo log.
- Page-level chains evict deleted rows from primary storage.
Solution: undo log for both pages and rows

- Old row versions form chains in undo log.
- Page-level chains evict deleted rows from primary storage.
- Update only indexes with changed values.
Huge WAL traffic.
- Huge WAL traffic.
- Problems with parallel apply.
- Huge WAL traffic.
- Problems with parallel apply.
- Not suitable for multi-master replication.
Solution: row-level WAL

- Very compact.

- Apply can be parallelized.

- Suitable for multi-master (row-level conflicts, not block-level).

- Recovery needs structurally consistent checkpoints.
Solution: row-level WAL

- Very compact.
- Apply can be parallelized.
Solution: row-level WAL

- Very compact.
- Apply can be parallelized.
- Suitable for multimaster (row-level conflicts, not block-level).
Solution: row-level WAL

- Very compact.
- Apply can be parallelized.
- Suitable for multimaster (row-level conflicts, not block-level).
- Recovery needs structurally consistent checkpoints.
Row-level WAL based multymaster
Copy-on-write checkpoints (1/4)
Copy-on-write checkpoints (2/4)
Copy-on-write checkpoints (3/4)
Copy-on-write checkpoints (4/4)

```
  1*
  /  \  \
 2    3*
  |    |   \\
 5    7*
  |    |   \\
2 4 5 6 Disk 7* 3* 1*
```

Alexander Korotkov

Solving PostgreSQL wicked problems
What do we need from PostgreSQL extendability?

- Extended table AM.

- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.
- Snapshot hooks.
- Some other miscellaneous hooks

Total 1K lines patch to PostgreSQL Core
What do we need from PostgreSQL extendability?

- Extended table AM.
- Custom toast handlers.
Extended table AM.
Custom toast handlers.
Custom row identifiers.
What do we need from PostgreSQL extendability?

- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
What do we need from PostgreSQL extendability?

- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkerpointer hooks.
What do we need from PostgreSQL extendability?

- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.

- Snapshot hooks.
What do we need from PostgreSQL extendability?

- Extended table AM.
- Custom toast handlers.
- Custom row identifiers.
- Custom error cleanup.
- Recovery & checkpointer hooks.

- Snapshot hooks.
- Some other miscellaneous hooks
  total 1K lines patch to PostgreSQL Core
OrioleDB = PostgreSQL redesign

PostgreSQL

- Block-level WAL
- Buffer mapping
- Buffer locking
- Bloat-prone MVCC
  - Cumbersome block-level WAL replication

Oriole

- Row-level WAL
- Direct page links
- Lock-less access
- Undo log
  - Raft-based multimaster replication of row-level WAL
### OrioleDB’s answer to 10 wicked problems of PostgreSQL

<table>
<thead>
<tr>
<th>Problem name</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Wraparound</td>
<td>Native 64-bit transaction ids</td>
</tr>
<tr>
<td>2. Failover Will Probably Lose Data</td>
<td>Multimaster replication</td>
</tr>
<tr>
<td>3. Inefficient Replication That Spreads Corruption</td>
<td>Row-level replication</td>
</tr>
<tr>
<td>4. MVCC Garbage Frequently Painful</td>
<td>Non-persistent undo log</td>
</tr>
<tr>
<td>5. Process-Per-Connection = Pain at Scale</td>
<td>Migration to multithread model</td>
</tr>
<tr>
<td>6. Primary Key Index is a Space Hog</td>
<td>Index-organized tables</td>
</tr>
<tr>
<td>7. Major Version Upgrades Can Require Downtime</td>
<td>Multimaster + per-node upgrade</td>
</tr>
<tr>
<td>8. Somewhat Cumbersome Replication Setup</td>
<td>Simple setup of raft-based multimaster</td>
</tr>
<tr>
<td>9. Ridiculous No-Planner-Hints Dogma</td>
<td>In-core planner hints</td>
</tr>
<tr>
<td>10. No Block Compression</td>
<td>Block-level compression</td>
</tr>
</tbody>
</table>

* Scalability on modern hardware
Let’s do some benchmarks! 

https://gist.github.com/akorotkov/f5e98ba5805c42ee18bf945b30cc3d67
OrioleDB benchmark: read-only scalability

Read-only scalability test PostgreSQL vs OrioleDB
1 minute of pgbench script reading 9 random values of 100M

OrioleDB: 4X higher TPS!
OrioleDB benchmark: read-write scalability in-memory case

Read-write scalability test PostgreSQL vs OrioleDB
1 minute of pgbench TPC-B like transactions wrapped into stored procedure

OrioleDB: 3.5X higher TPS!
OrioleDB benchmark: read-write scalability
external storage case

pgbench -s 20000 -j $n -c $n -M prepared on odb-node02
mean of 3 3-minute runs with shared_buffers = 32GB(128GB), max_connections = 2500

OrioleDB: up to 50X higher TPS!
OrioleDB benchmark: read-write scalability

Intel Optane persistent memory

pgbench -s 20000 -j $n -c $n -M prepared -f read-write-proc.sql on node03
5-minute run with shared_buffers = 32GB, max_connections = 2500

OrioleDB: up to 50X higher TPS!

Alexander Korotkov
Solving PostgreSQL wicked problems
OrioleDB benchmark: write-amplification & bloat test: CPU

OrioleDB: 5X higher TPS! 2.3X less CPU/TPS!

Alexander Korotkov
OrioleDB benchmark: write-amplification & bloat test: IO

OrioleDB: 5X higher TPS! 22X less IO/TPS!

Alexander Korotkov
Solving PostgreSQL wicked problems
OrioleDB benchmark: write-amplification & bloat test: space

OrioleDB: no bloat!
OrioleDB benchmark: taxi workload (1/3): read

OrioleDB: 9X less read IOPS!

Alexander Korotkov
Solving PostgreSQL wicked problems 35 / 40
OrioleDB benchmark: taxi workload (2/3): write

OrioleDB: 4.5X less write IOPS!

Alexander Korotkov
OrioleDB benchmark: taxi workload (3/3): space

OrioleDB: 8X less space usage!

Alexander Korotkov
Solving PostgreSQL wicked problems
37 / 40
OrioleDB = Solution of wicked PostgreSQL problems + extraordinary performance
Roadmap

- Basic engine features ✓
- Table AM interface implementation ✓
- Data compression ✓
- Undo log ✓
- TOAST support ✓
- Parallel row-level replication ✓
- Partial and expression indexes ✓

Initial release

- GiST/GIN analogues
Release is scheduled for December 1st 2021;
https://github.com/orioledb/orioledb;
If you need more explanation, don’t hesitate to make pull requests.