Evolution of Fault Tolerance in PostgreSQL
MSc, Computer and Systems Eng @ Tallinn Technical University
BSc, Applied Mathematics @ Yildiz Technical University
(was on) Board of Directors @ PostgreSQL Europe
Organizer @ Prague PostgreSQL Meetup
Working with databases for 10+ years
(lives in) Prague
(from) Turkey
(is) New Mom
Agenda

- PostgreSQL Fault Tolerance: WAL
- Fault Types in Database Systems
- Transaction - Commit - Checkpoint
- Replication Methods in PostgreSQL
- Failover and Switchover
- Managing Timeline Issues: pg_rewind
- Synchronous Replication (synchronous_commit)
- Logical Decoding
- Backups
Fault Tolerance

A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails.
Fault Types in Database Systems

1. User application bugs
2. Administrator (human) errors
3. Database software failures
4. Operating system failures
5. Hardware failures (disk)
6. Network failures
7. Datacenter-level events
PostgreSQL is Robust!

- Transactions + Transaction Log = **Automatic Crash Recovery** (Software failure)
- **Data block checksums** (Disk or file system faults)
- Multiple backup mechanisms (**Full PITR**)
- Low-level diagnostic tools (*pageinspect*)
- **Database replication** is supported **natively**.
  - Physical replication (page corruption can propagate to standbys)
  - Logical replication
Write ahead logging mechanism is the main fault tolerance system for PostgreSQL which ensures durability of any db changes.
Database changes themselves are not written to data files on disk at transaction commit. Writes to data files are done sometime later by the background writer or checkpoint on a server.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Read</th>
<th>Nonrepeatable Read</th>
<th>Phantom Read</th>
<th>Serialization Anomaly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>Allowed, but not in PG</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read committed</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Allowed, but not in PG</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializable</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

Standard SQL Transaction Isolation Levels
Checkpoint

Crash recovery replays the WAL, but from what point does it start to recover?
Database replication is the term we use to describe the technology used to maintain a copy of a set of data on a remote system.
Replication History

- **PostgreSQL 7.x**
  - Replication should not be part of core Postgres
  - 3rd party replication (usually trigger-based)

- **PostgreSQL 8.0**
  - Point-In-Time Recovery (WAL)

- **PostgreSQL 9.0**
  - Streaming Replication (physical)

- **PostgreSQL 9.4**
  - Logical Decoding (changeset extraction)

- **PostgreSQL 10+**
  - Logical Streaming Replication
Physical Replication

WAL over network from master to standby

- Streaming changes: using internal protocol (sender and receiver processes)
- Sending files: scp, rsync, ftp
### WAL and Replication

<table>
<thead>
<tr>
<th>WAL level</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>minimal</td>
<td>crash recovery</td>
</tr>
<tr>
<td>replica (default at PG12)</td>
<td>physical replication file-based archiving</td>
</tr>
<tr>
<td>logical</td>
<td>logical replication</td>
</tr>
</tbody>
</table>

**wal_level** parameter determines how much information is written to the **WAL**.
Failover and Switchover

Failover

Switchover
Timelines provide protection from connecting to the wrong upstream after promotion (failover, switchover).
There are outstanding changes in the old master
Timeline increase represents new history of changes
Changes from the old timeline can't be replayed on the servers that switched to new timeline
The old master can't follow the new master
Switchover

- There are no outstanding changes in the old master
- Timeline increase represents new history of changes
- The old master can become standby for the new master
pg_rewind (9.5+)

- Outstanding changes are removed using data from the new master
- The old master can follow the new master
Synchronous replication guarantees that data is written to at least two nodes before the user or application is told that a transaction has committed.
How Commit is Replicated?

Commit

Executor

Heap

WAL

WAL sender

WAL receiver

Executor

Heap

WAL
synchronous_commit = off
synchronous_commit = local
synchronous_commit = remote_write
synchronous_commit = on

Master

Commit

Standby
synchronous_commit = remote_apply
Logical replication allows us to stream logical data changes between two nodes.
Logical Decoding

- Since PostgreSQL 9.4
- Extracts information from Write-Ahead-Log into logical changes (INSERT/UPDATE/DELETE)
- Committed transactions
- Per row and commit ordered
- No write amplification
- C API for output plugin
- Does not decode DDL
- Streaming (WalSender) Interface
Logical Streaming Replication

Diagram:
- **Upstream** database
- **WAL** (Write-Ahead Logging)
- **Sender** process
- **Apply** mechanism
- **Downstream** database
- Data flow: users → upstream → WAL → sender → apply → downstream → WAL
Logical Streaming Replication

- PostgreSQL 10+
- Built on logical decoding
- Supports synchronous replication
- Publish Subscribe Model
  - CREATE PUBLICATION
  - CREATE SUBSCRIPTION
- Based on pglogical extension
Backups

- Replication alone is not enough
- You might need PITR
- How did backups evolve?
- Restore and Crash Recovery
- pg_basebackup
- Management of backups
Conclusion

What will be the next big leap of fault tolerance?
Questions?

Huge Thanks!