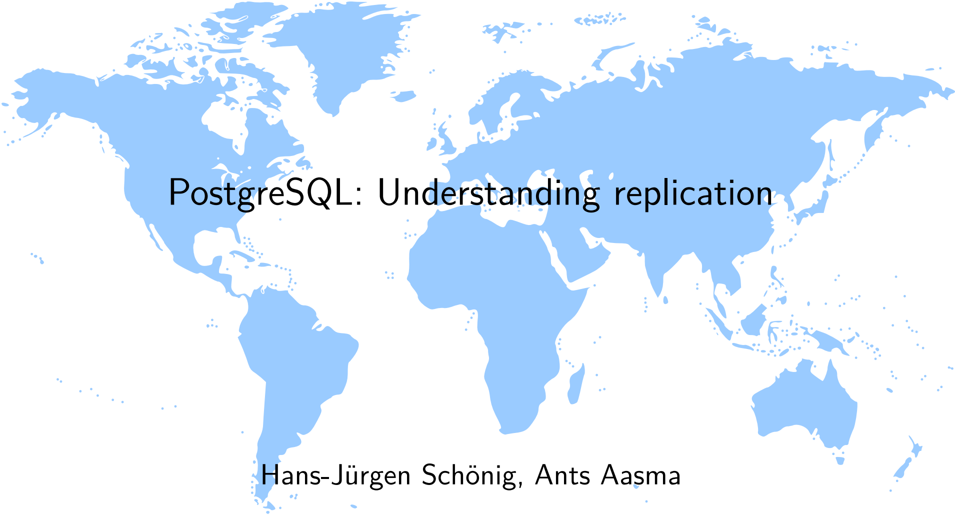

PostgreSQL: Understanding replication

Hans-Jürgen Schönig, Ants Aasma

www.cybertec.at

Welcome to PostgreSQL replication

What you will learn

- How PostgreSQL writes data
- What the transaction log does
- How to set up streaming replication
- Managing conflicts
- Monitoring replication
- More advanced techniques

How PostgreSQL writes data

Writing a row of data

- Understanding how PostgreSQL writes data is key to understanding replication
- Vital to understand PITR
- A lot of potential to tune the system

Write the log first (1)

- It is not possible to send data to a data table directly.
- What if the system crashes during a write?
- A data file could end up with broken data at potentially unknown positions
- Corruption is not an option

Write the log first (2)

- Data goes to the xlog (= WAL) first
- WAL is short for "Write Ahead Log"
- IMPORTANT: The xlog DOES NOT contain SQL
- It contains BINARY changes

The xlog

- The xlog consists of a set of 16 MB files
- The xlog consists of various types of records (heap changes, btree changes, etc.)
- It has to be flushed to disk on commit to achieve durability

Expert tip: Debugging the xlog

- Change WAL_DEBUG in src/include/pg_config_manual.h
- Recompile PostgreSQL

NOTE: This is not for normal use but just for training purposes

Enabling wal_debug

```
test=# SET wal_debug TO on;
SET
test=# SET client_min_messages TO debug;
SET
```

- Every change will go to the screen now
- It helps to understand how PostgreSQL works
- Apart from debugging: The practical purpose is limited

How does it impact replication

- The xlog has all the changes needed and can therefore be used for replication.
- Copying data files is not enough to achieve a consistent view of the data
- It has some implications related to base backups

Setting up streaming replication

The basic process

- S: Install PostgreSQL on the slave (no initdb)
- M: Adapt postgresql.conf
- M: Adapt pg_hba.conf
- M: Restart PostgreSQL
- S: Pull a base backup
- S: Start the slave

Changing postgresql.conf

- wal_level: Ensure that there is enough xlog generated by the master (recovering a server needs more xlog than just simple crash-safety)
- max_wal_senders: When a slave is streaming, connects to the master and fetches xlog. A base backup will also need 1 / 2 wal_senders
- hot_standby: This is not needed because it is ignored on the master but it saves some work on the slave later on

Changing pg_hba.conf

- Rules for replication have to be added.
- Note that "all" databases does not include replication
- A separate rule has to be added, which explicitly states "replication" in the second column
- Replication rules work just like any other pg_hba.conf rule
- Remember: The first line matching rules

Restarting PostgreSQL

- To activate those settings in postgresql.conf the master has to be restarted.
- If only pg_hba.conf is changed, a simple SIGHUP (pg_ctl reload) is enough.

Using pg_basebackup (1)

- pg_basebackup will fetch a copy of the data from the master
- While pg_basebackup is running, the master is fully operational (no downtime needed)
- pg_basebackup connects through a database connection and copies all data files as they are
- In most cases this does not create a consistent backup
- The xlog is needed to “repair” the base backup (this is exactly what happens during xlog replay anyway)

Using pg_basebackup (2)

```
pg_basebackup -h master.com -D /slave \
    --xlog-method=stream --checkpoint=fast -R
```

xlog-method: Self-contained backups

- By default a base backup is not self-contained.
- The database does not start up without additional xlog.
- This is fine for Point-In-Time-Recovery because there is an archive around.
- For streaming it can be a problem.
- –xlog-method=stream opens a second connection to fetch xlog during the base backup

checkpoint=fast: Instant backups

- By default pg_basebackup starts as soon as the master checkpoints.
- This can take a while.
- –checkpoint=fast makes the master check instantly.
- In case of a small backup an instant checkpoint speeds things up.

-R: Generating a config file

- For a simple streaming setup all PostgreSQL has to know is already passed to pg_basebackup (host, port, etc.).
- -R automatically generates a recovery.conf file, which is quite ok in most cases.

Backup throttling

- –max-rate=RATE: maximum transfer rate to transfer data directory (in kB/s, or use suffix "k" or "M")
- If your master is weak a pg_basebackup running at full speed can lead to high response times and disk wait.
- Slowing down the backup can help to make sure the master stays responsive.

Adjusting recovery.conf

- A basic setup needs:
  - primary_conninfo: A connect string pointing to the master server
  - standby_mode = on: Tells the system to stream instantly
- Additional configuration parameters are available

Starting up the slave

- Make sure the slave has connected to the master
- Make sure it has reached a consistent state
- Check for wal_sender and wal_receiver processes

Promoting a slave to master

- Promoting a slave to a master is easy:

```
pg_ctl -D ... promote
```

- After promotion recovery.conf will be renamed to recovery.done

One word about security

- So far replication has been done as superuser
- This is not necessary
- Creating a user, which can do just replication makes sense

```
CREATE ROLE foo ... REPLICATION ... NOSUPERUSER;
```

Converting master-to-slave

- New binary backup (pg_basebackup, rsync, ...)
- pg_rewind
- Rolls back changes based on WAL
- Requires that hint bits are logged (wal_log_hints)

Monitoring replication

Simple checks

- The most basic and most simplistic check is to check for
  - wal_sender (on the master)
  - wal_receiver (on the slave)
- Without those processes the party is over

More detailed analysis

- pg_stat_replication contains a lot of information
- Make sure an entry for each slave is there
- Check for replication lag

Checking for replication lag

- A sustained lag is not a good idea.
- The distance between the sender and the receiver can be measured in bytes

```
SELECT client_addr,
    pg_xlog_location_diff(pg_current_xlog_location(),
        sent_location)
FROM pg_stat_replication;
```

- In asynchronous replication the replication lag can vary dramatically (for example during CREATE INDEX, etc.)

Creating large clusters

Handling more than 2 nodes

- A simple 2 node cluster is easy.
- In case of more than 2 servers, life is a bit harder.
- If you have two slaves and the master fails: Who is going to be the new master?
  - Unless you want to resync all your data, you should better elect the server containing most of the data already
  - Comparing xlog positions is necessary

Timeline issues

- When a slave is promoted the timeline ID is incremented
- Master and slave have to be in the same timeline
- In case of two servers it is important to connect one server to the second one first and do the promotion AFTERWARDS.
- This ensures that the timeline switch is already replicated from the new master to the surviving slave.

Cascading slaves

- Slaves can be connected to slaves
- Cascading can make sense to reduce bandwidth requirements
- Cascading can take load from the master
- Use pg_basebackup to fetch data from a slave as if it was a master

Conflicts

How conflicts happen

- During replication conflicts can happen
- Example: The master might want to remove a row still visible to a reading transaction on the slave

What happens during a conflict

- PostgreSQL will terminate a database connection after some time
  - max_standby_archive_delay = 30s
  - max_standby_streaming_delay = 30s
- Those settings define the maximum time the slave waits during replay before replay is resumed.
- In rare cases a connection might be aborted quite soon.

Reducing conflicts

- Conflicts can be reduced nicely by setting hot_standby_feedback.
  - The slave will send its oldest transaction ID to tell the master that cleanup has to be deferred.

Making replication more reliable

What happens if a slave reboots?

- If a slave is gone for too long, the master might recycle its transaction log
- The slave needs a full history of the xlog
- Setting wal_keep_segments on the master helps to prevent the master from recycling transaction log too early
- I recommend to always use wal_keep_segments to make sure that a slave can be started after a pg_basebackup

Making use of replication slots

- Replication slots have been added in PostgreSQL 9.4
- There are two types of replication slots:
  - Physical replication slots (for streaming)
  - Logical replication slots (for logical decoding)

Configuring replication slots

- Change max_replication_slots and restart the master
- Run ...

```
test=# SELECT *
    FROM pg_create_physical_replication_slot('some_name');
 slot_name | xlog_position
-----------+---------------
 some_name |
(1 row)
```

- Add this replication slot to primary_slot_name on the slave:

```
primary_slot_name = 'some_name'
```

- The master will ensure that xlog is only recycled when it has been consumed by the slave.

A word of caution

- If a slave is removed make sure the replication slot is dropped.
- Otherwise the master might run out of disk space.
- NEVER use replication slots without monitoring the size of the xlog on the sender.

Key advantages of replication slots

- The difference between master and slave can be arbitrary.
- During bulk load or CREATE INDEX this can be essential.
- It can help to overcome the problems caused by slow networks.
- It can help to avoid resyncs.

Moving to synchronous replication

Synchronous vs. asynchronous

- Asynchronous replication: Commits on the slave can happen long after the commit on the master.
- Synchronous replication: A transaction has to be written to a second server.
- Synchronous replication potentially adds some network latency to the scenery

- During normal operations the application_name setting can be used to assign a name to a database connection.
- In case of synchronous replication this variable is used to determine synchronous candidates.

Configuring synchronous replication:

- Master:
  - add names to synchronous_standby_names
- Slave:
  - add an application_name to your connect string in primary_conninfo

Fail safety

- Synchronous replication needs 2 active servers
- If no two servers are left, replication will wait until a second server is available.
- Use AT LEAST 3 servers for synchronous replication to avoid risk.

Logical replication

Before 9.4

- Trigger based solutions

Logical replication overview

- Extracts transactional changesets from transaction logs
- Output plugins to convert changes to a useful format.
- Example output formats: SQL, JSON
- BDR project for multi-master replication.

Logical replication demo

- Lets receeive some data using pg_recvlogical