Streaming replication in practice.

PgConf.Russia 2016, Moscow

Lesovsky Alexey

lesovsky@pgco.me

PostgreSQL-Consulting.com

Agenda.

Working stuff: http://goo.gl/Yy4UzH

I. How streaming replication works.

II. Replication setup.

Practice.

III. Monitoring and maintenance.

IV. Possible problems.

V. Switchiver и Failover.

Practice.

Part I. How replication works.

What is replication and it kinds?

Write Ahead Log. REDO and REDO realization in PostgreSQL.

Common replication architecture in PostgreSQL.

Description of processes that involved in replication.

What is replication.

Synchronize objects

Objects changes are moving to each others.

Replication maybe physical and logical.

Logical replication.

Pros:

- Works between different major versions and architectures.
- Allow to replicate tables and tables sets.

Cons:

- Hard to implement synchronous replication.
- CPU overhead (triggers, text conversions, etc).

Examples:

- Slony, Londiste (Skytools), Bucardo, Pglogical.

Physical replication.

Pros:

- Resource usage minimal overhead.
- Easy setup, usage and maintenace.

Cons:

- Standbys are only read-only.
- Can't work with different versions and architectures.
- Can't replicate tables and tables sets.

Write Ahead Log. REDO.

Commit all changes in database (Durability in ACID).

Flush all data from REDO buffers at COMMIT.

REDO log has history of all changes in database.

Any changes in database, written into REDO.

REDO log usage:

- in crash recovery;
- backup and Point In Time Recovery;
- replication.

Write Ahead Log. REDO implementation in PostgreSQL.

In PostgreSQL, REDO also known as Write Ahead Log (WAL).

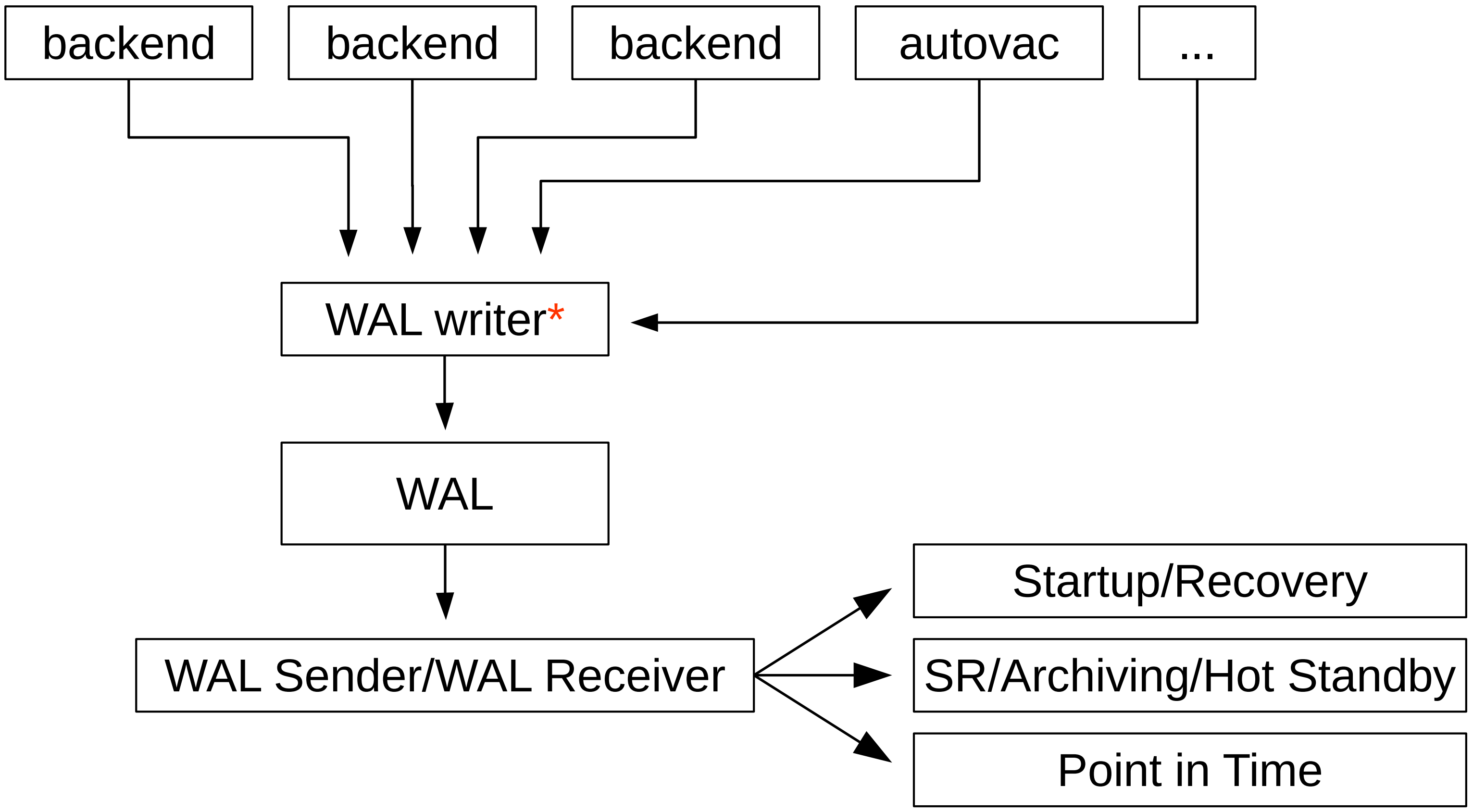
WAL guaranties that changes are commited before data will changed.

How it guarantied:

LSN (log sequence number) — record location (position) inside WAL;

- Any page is marked with LSN of last record that touched the page;
- Before page is written to disk, bufmgr must check that the WAL flushed to specified LSN.

Common view.



* - может отсутствовать

Startup process.

Main startup process task is run the database.

In standby mode it initializes infinite replay loop.

Read recovery.conf at REDO start.

REDO:

- read segments from pg_xlog/archive;
- start wal receiver and reading XLOG from upstream server.

When consistency point reached (min recovery ending location) allow connections and starts checkpointer/bgwriter.

Processing all others parameters from recovery.conf.

More details see in StartupXLOG() function.

WAL Sender process.

For any client postmaster runs dedicated process — backend.

WAL sender is backend too (it has am_walsender flag).

This backend runs exec_replication_command().

exec_replication_command() can do various things:

- create/remove replication slots;
- start basebackup;
- start physical/logical replication.

In last case, backend sends XLOG segments to the client.

Or sleeps when no new XLOG.

WAL Receiver process.

Startup process checks XLOG sources.

Startup process init startup of WAL receivers.

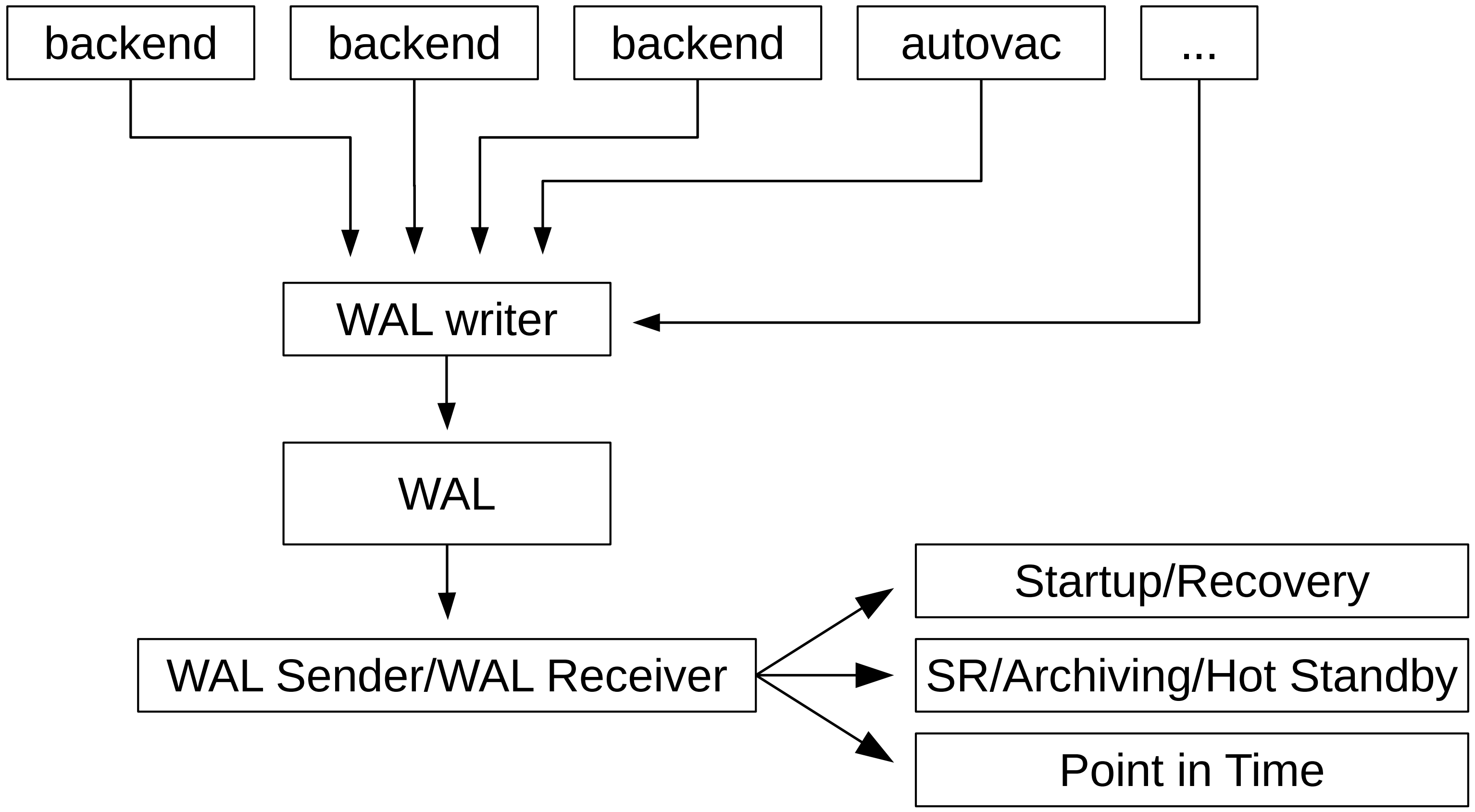
Need recovery.conf with primary_conninfo.

WAL receiver:

- check start location for transfer XLOG;
- connects to master and sends start position;
- receive XLOG and write it to disk;
- update variable in shared memory (WalRcv→receivedUpto);
- sends statistics + feedback.

Startup process uses the variable and replay WAL to this location.

Questions?


backend
backend
backend
autovac
...
WAL writer
WAL
WAL Sender/WAL Receiver
Startup/Recovery
SR/Archiving/Hot Standby
Point in Time

Part II. Replication setup.

Setup options.

Prepare master for replication.

Tools and utilities.

Start replication and verify results.

Specific options.

Setup options.

Synchronous/Asynchronous replication.

Cascade configurations.

Uni-directional/Bi-directional.

Common logic.

Prepare the master.

Copy the DATADIR.

Prepare standbys.

Start standby.

Result verify.

Master setup.

Dedicated user for a replication.

Edit the postgresql.conf.

Edit the pg_hba.conf.

Create replication slots (if required).

Master setup.

Dedicated user for a replication.

- CREATE ROLE ... WITH LOGIN REPLICATION PASSWORD '…';

Edit the postgresql.conf.

Edit the pg_hba.conf.

Create replication slots (if required).

Master setup.

Dedicated user for a replication.

Edit the postgresql.conf.

- wal_level = hot_standby
- max_wal_senders > 0
- Restart the PostgreSQL

Edit the pg_hba.conf.

Create replication slots (if required).

Master setup.

Dedicated user for a replication.

Edit the postgresql.conf.

Edit the pg_hba.conf.

- host replication username client_addr/mask authtype
- host replication replica 10.1.0.99/32 md5
- pg_reload_conf()

Create replication slots (if required).

Master setup.

Dedicated user for a replication.

Edit the postgresql.conf.

Edit the pg_hba.conf.

Create replication slots (if required).

- max_replication_slots > 0
- pg_create_physical_replication_slot('slotname');
- primary_slot_name = 'slotname' (recovery.conf)

Copy the DATADIR.

pg_basebackup (since 9.1)

-h, --host=...; -p, --port=...; -U, --username=...; -d, --dbname=...; -D, --pgdata=...

-c, --checkpoint=fast | spread	
-X, --xlog-method=fetch | stream	– stream c 9.2
-R, --write-recovery-conf	– c 9.3
-r, --max-rate=…	– c 9.4
--xlogdir=…	– c 9.4
-T, --tablespace-mapping=olddir=newdir	– c 9.4
-P, --progress	

```
pg_basebackup -P -R -X stream -c fast -h 127.0.0.1 -U replica -D /pgdb
```

Copy the DATADIR. pg_basebackup alternatives.

Copy with cp, scp, tar, rsync...

Snapshots:

- ZFS send/receive;
- LVM + dd.

pg_start_backup() + pg_stop_backup().

Standby setup.

Configuration files should be the same.

Configuration files:

- postgresql.conf;
- recovery.conf.

Standby setup.

Configuration files should be the same:

- Why?
- How?

Configuration files:

- postgresql.conf;
- recovery.conf.

Standby setup.

Configuration files should be the same.

Configuration files (postgresql.conf):

- hot_standby = on;
- max_standby_streaming_delay;
- wal_receiver_status_interval;
- hot_standby_feedback;
- wal_receiver_timeout;

Standby setup.

Configuration files should be the same.

Configuration files (recovery.conf):

- primary_conninfo = 'host=… port=…'
- standby_mode = on
- primary_slot_name = 'slotname'
- trigger_file = '...'
- recovery_min_apply_delay.

Standby start.

pg_ctl — PostgreSQL native utility.

pg_ctlcluster — pg_ctl perl wrapper in Debian/Ubuntu.

sysvinit, upstart, openrc, systemd…

Check results.

wal sender and wal receiver processes.

Check postgres log.

Simple connection with psql.

pg_stat_replication view.

Specific options.

DATADIR, configs and Debian-based vs. RHEL-based.

pg_ctlcluster and «unable to connect» errors.

To less processes in «ps» output.

Resume. Practical purpose of the replication.

Read scalability.

Full text search.

OLAP.

Standby is not a backup.

Questions.

Replication setup. Practice.

root password: pgconf2016

# su - postgres	– working under postgres account.
$ ps auxf	– what we have?
$ pwd	– where we are?
$ ls -l 9.5	

Master setup.

$ vi 9.5/data/postgresql.conf

• listen_addresses = '*'	– Listen on all interfaces.
• wal_level = hot_standby	– Set WAL verbose level.
• max_wal_senders = 4	– Limiting walsenders.
• hot_standby = on	– Allow read-only queries on standby.

Master setup.

```
$ psql                                      – Creare dedicated user.
    CREATE ROLE replica WITH LOGIN REPLICATION PASSWORD 'rep123';
$ vi .pgpass                                – Setup password file.
    *:*:*:replica:rep123
$ chmod 600 .pgpass
$ vi 9.5/data/pg_hba.conf                   – Add auth rules for replication user.
    host   replication   replica    127.0.0.1/32      md5
$ pg_ctl -D 9.5/data/ -m fast restart       – Apply the changes.
```

Create the standby.

```
$ pg_basebackup -P -R -c fast -X stream -h 127.0.0.1 -U replica -D 9.5/replica
```

- -c fast — do the force checkpoint.
- -X stream — copy new XLOG through dedicated connection.
- -R — create minimal recovery.conf

```
$ vi 9.5/replica/postgresql.conf
```
– edit port number.

```
    port = 5433
```

```
$ pg_ctl -D 9.5/replica/ start
```
– start this standby.

Check result.

$ ps auxf	– wal sender/receiver process.
$ psql -p 5433	– check the status on the standby.
select pg_is_in_recovery();	– standby nust be in recovery mode.
$ psql	– check the status on the master.
select * from pg_stat_replication ;	– check statistics from standby.

Questions.

Yes, replication is ready.

Part III. Monitoring and maintenance.

«Setup and forget» - this is about PostgreSQL streaming replication.

Monitoring:

- Internal statistics;
- auxiliary functions;
- queries examples.

Maintenance:

- add or remove standbys;
- pause replication;
- add or remove slots.

Monitoring.

System views:

- pg_stat_replication
- pg_stat_replication_slots

Monitoring. Auxiliary functions.

pg_is_in_recovery()

pg_current_xlog_location()

pg_last_xact_replay_timestamp()

pg_last_xlog_receive_location()

pg_last_xlog_replay_location()

pg_xlog_location_diff()

Monitoring. Queries examples.

Replication monitoring on master.

```
select
  pid, client_addr,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),sent_location)) as
pending_xlog,
  pg_size_pretty(pg_xlog_location_diff(sent_location,write_location)) as write,
  pg_size_pretty(pg_xlog_location_diff(write_location,flush_location)) as flush,
  pg_size_pretty(pg_xlog_location_diff(flush_location,replay_location)) as replay,
  pg_size_pretty(pg_xlog_location_diff(pg_current_xlog_location(),replay_location)) as
total_lag
from pg_stat_replication;
```

```
  pid  |  client_addr   | pending_xlog |  write    |  flush   |  replay     | total_lag
-------+----------------+--------------+-----------+----------+-------------+------------
 21015 | 127.0.0.1      | 0 bytes      | 0 bytes   | 0 bytes | 48 bytes    | 48 bytes
  2067 | 192.168.200.4 | 12 GB        | 30 MB     | 0 bytes | 156 kB      | 12 GB
 18635 | 192.168.100.2 | 0 bytes      | 48 bytes | 0 bytes | 590 MB      | 590 MB
```

Monitoring.

On standby:

pg_current_xlog_location() → pg_last_xlog_receive_location()

WAL amount:

- SELECT pg_xlog_location_diff(pg_current_xlog_location, '0/0');

Lag in seconds:

- SELECT now() - pg_last_xact_replay_timestamp();

Maintenance.

Add new standbys or removing existing standbys.

Temporary pause replication.

Maintenance.

Add new standbys or removing existing standbys.

- max_wal_senders
- max_replication_slots
- pg_create_physical_replication_slot()
- pg_drop_replication_slot()

Temporary pause replication.

Maintenance.

Add new standbys or removing existing standbys.

Temporary pause replication:

- pg_is_xlog_replay_paused()
- pg_xlog_replay_pause()
- pg_xlog_replay_resume()

Questions.

Part IV. Problems.

Replication lag.

Replication stopping.

Disk and network problems.

100% disk usage.

Recovery conflicts.

Tables and indexes bloat.

pg_xlog/ bloat.

Replication lag.

Symptoms:

- Data between standby and master are differ.

Causes:

- Long queries on standby, much writes on master;
- Hardware issues.

Solutions:

- Application optimizations.

Networking and Storage.

Network lag:

- full_page_writes = off;
- ssh tunnels with compression.

Storage lag:

- full_page_writes =off;
- filesystem barriers;
- writethrough/writeback;
- RAID BBU learning;
- ionice (only for cfq elevator).

Replication stopping.

Symptoms:

- Recovery process uses 100% CPU;
- Lag increasing.

Causes:

- Heavy update/delete, too many autovacuums.

Solutions:

- Increasing wal_keep_segments;
- Temporary disabling of full_page_writes;
- Set priorities with ionice and renice.

100% disk usage.

Causes:

- Replication slots and stopped standby → save XLOG segments;

Solutions:

- Remove the slot and use wal_keep_segments.

Dirty hack:

- Filesystem's reserved blocks percentage and tune2fs.

Recovery conflicts.

Why conflicts occurs:

- Autovacuum;
- XLOG replay.

Solutions:

- hot_standby_feedback = on;
- Increasing max_standby_streaming_delay.

Tables and indexes bloat.

Causes:

- Long transactions on a standby.

Solutions:

- pgstattuple;
- VACUUM FULL, pgcompacttable, pg_reorg...;

pg_xlog/ bloat on a standby.

Symptoms:

- Different size pg_xlog/ and amount of XLOG segments.

Solutions:

- Decreasing checkpoint_timeout;
- Decreasing checkpoint_completion_target.

Questions.

Part V. Switchover and Failover.

What is it?

For what is needed?

How to do it?

Prerequisites.

Switchover and Failover.

Purposes:

- Updates of software, operating system, or hardware.
- Hardware failures.

Switchover.

Run chekpoint on master.

Check replication lag.

Shutdown the master.

Remove recovery.conf and restart a standby.

Switchover.

Pros:

- Old master fast reuse;
- No lost transactions.

Cons:

- Warm cache after restart;
- pg_prewarm extension (since 9.4).

Failover.

Create trigger file

- recovery.conf: trigger_file = '…'
- Need restart after recovery.conf changes.

With pg_ctl:

- pg_ctl -D ... promote

Failover.

Pros:

- It's fast;
- Don't need a restart;
- Don't need a cache warm.

Cons:

- Lost transactions risk;
- Old master should be reinitialized (until 9.5).

Old master reuse.

Switchover:

- create recovery.conf and start.

Failover:

- reinit as standby (until 9.5);
- pg_rewind (since 9.5).
  - timeline must be differs between master and standby.
  - old master shut be shutdowned correctly.
  - but sometimes issues occurs.
  - pg_rewind --target-pgdata=9.5/main --source-server="host=10.0.0.1"

Questions.

Switchover. Practice.

```
$ vi 9.5/replica/postgresql.conf            – edit configuration before restart.
      port = 5432
$ mv 9.5/replica/recovery.conf /tmp/        – remove recovery.conf from DATADIR
$ psql
> CHECKPOINT;                               – reduce restart time.
$ pg_ctl -D 9.5/data -m fast stop           – shutdown the master.
$ pg_ctl -D 9.5/replica -m fast restart     – promote new master.
$ tail -f 9.5/replica/pg_log/postgresql-Wed.log
$ ps auxf
```

Switchover. Reuse old master.

```
$ vi 9.5/data/postgresql.conf              – edit config.
        port = 5433
$ mv /tmp/recovery.conf 9.5/data/          – create recovery.conf.
$ pg_ctl -D 9.5/data start                 – start.
$ ps auxf                                  – check.
```

Failover. Prepare.

```
$ vi 9.5/data/postgresql.conf
$ vi 9.5/replica/postgresql.conf
```

- wal_log_hints = on – this options required for pg_rewind
- wal_keep_segments = 32

```
$ pg_ctl -D 9.5/data -m fast restart
$ pg_ctl -D 9.5/replica -m fast restart
```

Failover.

$ pg_ctl -D 9.5/replica -m immediate stop	– «crash» a master.
$ pg_ctl -D 9.5/data promote	– promote a standby.
$ psql -p 5433	– «doing the changes».
create database test;	

Failover. Reuse the old master.

```
$ pg_ctl -D 9.5/replica start
$ pg_ctl -D 9.5/replica stop
$ pg_rewind -D 9.5/replica --source-server="host=127.0.0.1 port=5433"
$ vi 9.5/replica/postgresql.conf
        port = 5432
$ mv 9.5/replica/recovery.done 9.5/replica/recovery.conf
$ vi 9.5/replica/recovery.conf
        port = 5433
$ pg_ctl -D 9.5/replica start
$ ps auxf
```

Questions.

Thanks.

Alexey Lesovsky, PostgreSQL Consulting.

lesovsky@pgco.me