PostgreSQL 9.4 - streaming replication

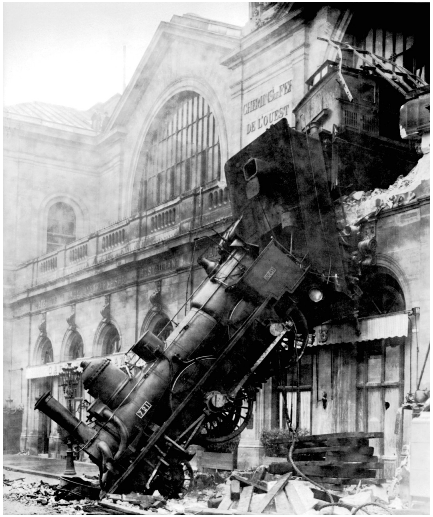
Federico Campoli

27 Nov 2015

Table of contents

Table of contents

Crash, Recovery and their band of merry men



Montparnasse derailment

Crash, Recovery and their band of merry men

The word ACID is an acronym for Atomicity, Consistency, Isolation and Durability. An ACID compliant database ensures those rules are enforced at any time.

- Atomicity requires that each transaction be “all or nothing”
- The consistency property ensures that any transaction will bring the database from one valid state to another
- The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially
- The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

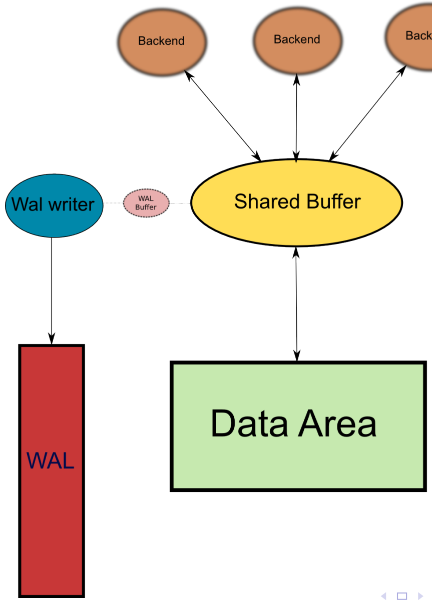
Source Wikipedia

Crash, Recovery and their band of merry men

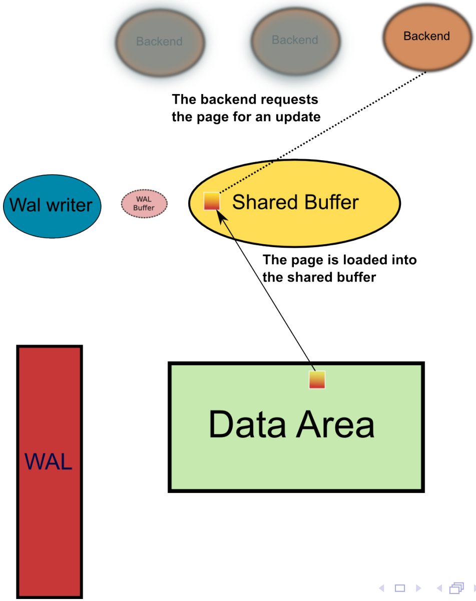
PostgreSQL implements the durability using the Write Ahead Logging.

When a page is updated in the volatile memory a so called xlog record is written on the write ahead log for the crash recovery.

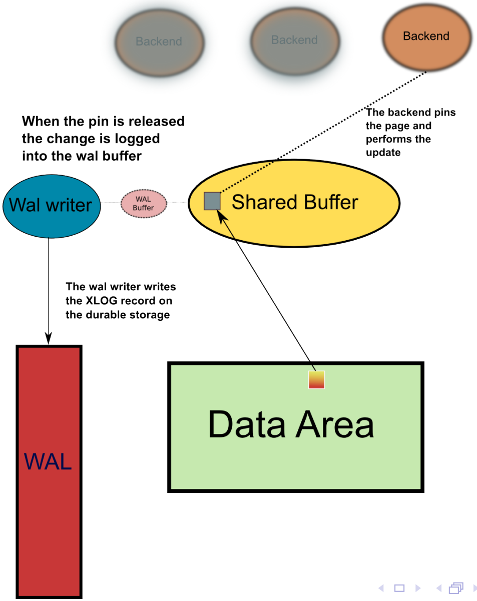
Crash, Recovery and their band of merry men



Crash, Recovery and their band of merry men


Backend
Backend
Backend
The backend requests the page for an update
Wal writer
WAL Buffer
Shared Buffer
The page is loaded into the shared buffer
Data Area
WAL

Crash, Recovery and their band of merry men


Backend
Backend
Backend
When the pin is released the change is logged into the wal buffer
The backend pins the page and performs the update
Wal writer
WAL Buffer
Shared Buffer
The wal writer writes the XLOG record on the durable storage
Data Area
WAL

Crash, Recovery and their band of merry men

- The WAL segments are stored in the directory $PGDATA/pg_xlog
- Each segment is usually 16 MB
- When the segment is full then PostgreSQL switches to another segment
- The number of segments is managed by PostgreSQL

Crash, Recovery and their band of merry men

- The page in memory which is updated but not yet written on the data area is called dirty
- The actual write happens either when the background writer processes the page or at the checkpoint
- The checkpoint frequency is controlled by the parameters checkpoint_timeout and checkpoint_segments

Crash, Recovery and their band of merry men

When the checkpoint happens

- All the dirty pages in the shared buffer are written to disk
- The control file is updated with the last recovery location
- The WAL files are recycled or removed

Crash, Recovery and their band of merry men

If the server crashes with dirty pages in memory

- At the startup the control file is accessed to get the last recovery location
- The WAL files are scanned and all the XLOG records are replayed
- A checkpoint is triggered at the end of the recovery

Table of contents

Point in Time and Recovery in space

Point in Time and Recovery in space

When the server switches to another wal file the old one becomes available for eviction or recycling at the next checkpoint.

Point in Time and Recovery in space

When the server switches to another wal file the old one becomes available for eviction or recycling at the next checkpoint.

If we save this file in another location and take an inconsistent copy of the data area, we can reconstruct the server physical copy.

Point in Time and Recovery in space

When the server switches to another wal file the old one becomes available for eviction or recycling at the next checkpoint.

If we save this file in another location and take an inconsistent copy of the data area, we can reconstruct the server physical copy.

So simple?

Point in Time and Recovery in space

When the server switches to another wal file the old one becomes available for eviction or recycling at the next checkpoint.

If we save this file in another location and take an inconsistent copy of the data area, we can reconstruct the server physical copy.

So simple?

Not exactly.

Point in Time and Recovery in space

The control file is constantly written and therefore is not a source of truth for the last checkpoint location.

The wal file does not contains the transaction commit status nor the vacuum operations.

Point in Time and Recovery in space

The control file is constantly written and therefore is not a source of truth for the last checkpoint location.

The wal file does not contains the transaction commit status nor the vacuum operations.

The configuration file needs some adjustments.

Point in Time and Recovery in space

The control file is constantly written and therefore is not a source of truth for the last checkpoint location.

The wal file does not contains the transaction commit status nor the vacuum operations.

The configuration file needs some adjustments.
Changing the following parameters requires a server restart.

- archive_mode set to 'on'

Point in Time and Recovery in space

The control file is constantly written and therefore is not a source of truth for the last checkpoint location.

The wal file does not contains the transaction commit status nor the vacuum operations.

The configuration file needs some adjustments.
Changing the following parameters requires a server restart.

- archive_mode set to 'on'
- wal_level set to archive, hot_standby or logical

Point in Time and Recovery in space

Changing archive_command requires only a server reload.

Point in Time and Recovery in space

```
archive_command = 'test ! -f /pg_archive/%f && cp %p /pg_archive/%f'
```

Each time a WAL is switched the archive command is executed to save the file.

Point in Time and Recovery in space

Start the backup with

```
postgres=# SELECT pg_start_backup('PITR', 't');
 pg_start_backup
-----------------
 0/3000028
(1 row)
```

The command issues a checkpoint and creates the file backup_label in the data area. In this file it's written the recovery WAL's start location.

```
START WAL LOCATION: 1/28000028 (file 000000010000000100000028)
CHECKPOINT LOCATION: 1/28000060
BACKUP METHOD: pg_start_backup
BACKUP FROM: master
START TIME: 2015-11-22 17:47:23 UTC
LABEL: PITR
```

Point in Time and Recovery in space

Save the running cluster's data area and all the tablespaces

- rsync
- copy
- tar
- cpio

Point in Time and Recovery in space

Tell the server the backup is complete with pg_stop_backup();

```
postgres=# SELECT pg_stop_backup();
NOTICE:  pg_stop_backup complete, all required WAL segments have been
     archived
 pg_stop_backup
----------------
 1/2C0000F0
(1 row)
```

The command deletes the backup_label and switches the current log file in order archive all the required segments.

Point in Time and Recovery in space

If a recovery is needed, we shall restore the data directory. Then, inside the data area, we must create a text file called recovery.conf.
The file is used to set the recovery strategy.

Point in Time and Recovery in space

If a recovery is needed, we shall restore the data directory. Then, inside the data area, we must create a text file called recovery.conf.
The file is used to set the recovery strategy.

```
restore_command = 'cp /pg_archive/%f %p'
```

This command does the opposite of the archive_command set previously. It's the copy command for restoring the archived WALs into the pg_xlog.

Point in Time and Recovery in space

recovery_target = 'immediate'
This parameter specifies that recovery should end as soon as a consistent state is reached, i.e. as early as possible. When restoring from an online backup, this means the point where taking the backup ended.

Point in Time and Recovery in space

recovery_target_time (timestamp)

This parameter specifies the time stamp up to which recovery will proceed. The precise stopping point is also influenced by recovery_target_inclusive.

Point in Time and Recovery in space

recovery_target_inclusive (boolean)

Specifies whether to stop just after the specified recovery target (true), or just before the recovery target (false). Applies when either recovery_target_time or recovery_target_xid is specified. This setting controls whether transactions having exactly the target commit time or ID, respectively, will be included in the recovery. Default is true.

Point in Time and Recovery in space

The PITR enforces the disaster recovery.

Point in Time and Recovery in space

The PITR enforces the disaster recovery.
Which comes very handy if, for example, somebody drops a table by accident.

Point in Time and Recovery in space

The PITR enforces the disaster recovery.
Which comes very handy if, for example, somebody drops a table by accident.
Alongside with this

Copyright Tim Avatar Bartel

Table of contents

A heap of broken WALs

A heap of broken WALs

As soon as the recovery target is reached the server becomes a standalone instance generating a new timeline.

The recovery.conf can also be configured in order to set the server in continuous recovery.

In this configuration we are talking of a standby server.

The standby server helps to enforce the high availability because replays the master's changes in almost real time.

The standby server can be warm or hot standby. The latter configuration allows the read only queries.

A heap of broken WALs

Standby server's minimal recovery.conf

```
standby_mode = 'on'
restore_command = 'cp /pg_archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /pg_archive %r'
```

A heap of broken WALs

Slave's hot standby configuration

```
hot_standby='on'
max_standby_archive_delay='30s'
```

A heap of broken WALs

Using the wal shipping for the standby have some limitations.

- is not realtime

A heap of broken WALs

Using the wal shipping for the standby have some limitations.

- is not realtime
- the network can be an issue

A heap of broken WALs

Using the wal shipping for the standby have some limitations.

- is not realtime
- the network can be an issue
- archive corruption leads to a broken standby server

A heap of broken WALs

Using the wal shipping for the standby have some limitations.

- is not realtime
- the network can be an issue
- archive corruption leads to a broken standby server
- the WAL files are stored in the slave's archive and then copied into to the pg_xlog

Table of contents

And now for something completely different

And now for something completely different

PostgreSQL 9.0 introduced the streaming replication which is physical block replication over a database connection.

- the WALs are streamed using a database connection in almost realtime

And now for something completely different

PostgreSQL 9.0 introduced the streaming replication which is physical block replication over a database connection.

- the WALs are streamed using a database connection in almost realtime
- the WALs are saved in the pg_xlog

And now for something completely different

PostgreSQL 9.0 introduced the streaming replication which is physical block replication over a database connection.

- the WALs are streamed using a database connection in almost realtime
- the WALs are saved in the pg_xlog
- it supports the synchronous slaves

And now for something completely different

PostgreSQL 9.0 introduced the streaming replication which is physical block replication over a database connection.

- the WALs are streamed using a database connection in almost realtime
- the WALs are saved in the pg_xlog
- it supports the synchronous slaves
- replication slots simplifies the streaming replication only slaves

And now for something completely different

On the master add an user with the replication privilege

```
CREATE ROLE usr_replication WITH REPLICATION PASSWORD 'EiHohG2z' LOGIN;
```

Update the master's postgresql.conf

```
max_wal_senders = 2 #requires restart
wal_level = hot_standby #requires restart
wal_keep_segments = 32
```

And now for something completely different

Add an entry in the master's pg_hba.conf for the “virtual” database replication

```
host replication usr_replication 192.168.0.20/22 md5
```

And now for something completely different

Add the connection info the slave's recovery.conf

```
primary_conninfo='dbname=replication user=usr_replication
                  host=pg_master password=EiHohG2z port=5432'
```

And now for something completely different

Restarting the slave it will reply the WAL files from the archive like a normal PITR/standby.

Only when there are no more WALs available to restore the slave will connect to the master using the connection string in primary_conninfo.

If the connection succeeds the slave will start streaming the WAL files from the master's pg_xlog directly into its own pg_xlog.

And now for something completely different

A single machine master/slave setup

And now for something completely different

A single machine master/slave setup
Master setup

```
ALTER SYSTEM SET archive_mode ='on';
ALTER SYSTEM SET wal_level ='hot_standby';
ALTER SYSTEM SET archive_command ='test ! -f /pg_archive/%f && cp %p /
    pg_archive/%f';
ALTER SYSTEM SET max_wal_senders = '2';
ALTER SYSTEM SET wal_keep_segments = '32';
ALTER SYSTEM SET hot_standby = 'on';
ALTER SYSTEM SET listen_addresses = '*';
CREATE ROLE usr_replication WITH REPLICATION PASSWORD 'EiHohG2z' LOGIN;
```

Add the following entry in the IPv4 section of the master's pg_hba.conf

```
host replication usr_replication 127.0.0.1/32 md5
```

Restart the master for applying the changes.

And now for something completely different

Check that the WAL archive is working

```
postgres=# SELECT pg_switch_xlog();
 pg_switch_xlog
----------------
 0/16ACA00
(1 row)
```

```
 thedoctor@tardis:~$ ls -lh /pg_archive
total 16M
thedoctor thedoctor 16M Nov 26 06:40 000000010000000000000001
```

And now for something completely different

Start the backup on the master and copy the data area in another location

```
postgres=# SELECT pg_start_backup('PITR', 't');
 pg_start_backup
-----------------
 0/3000028
(1 row)
```

And now for something completely different

Start the backup on the master and copy the data area in another location

```
postgres=# SELECT pg_start_backup('PITR', 't');
 pg_start_backup
-----------------
 0/3000028
(1 row)
```

Copy the data area in a different directory

```
rsync --exclude "*postmaster.pid*" -va /pg_data/9.4/master/ \
   /pg_data/9.4/slave/

sent 70,760,203 bytes
received 14,819 bytes
47,183,348.00 bytes/sec
total size is 70,699,548  speedup is 1.00
```

And now for something completely different

Stop the backup on the master

```
postgres=# SELECT pg_stop_backup();
NOTICE:  pg_stop_backup complete, all required WAL segments have been
    archived
 pg_stop_backup
----------------
 0/3000128
(1 row)
```

And now for something completely different

In our example the slave needs a different port because is on the same machine. Change the parameter port to 5433 in

```
/pg_data/9.4/slave/postgresql.conf
```

Create the file recovery.conf in the slave's data area and add the configuration for the streaming replication

```
primary_conninfo='dbname=replication user=usr_replication
                  host=localhost password=EiHohG2z port=5432'
standby_mode = 'on'
restore_command = 'cp /pg_archive/%f %p'
archive_cleanup_command = 'pg_archivecleanup /pg_archive %r'
```

And now for something completely different

Start the slave

```
export PGDATA=/pg_data/9.4/slave
pg_ctl start
```

```
 thedoctor@tardis:/pg_data/9.4$ pg_ctl start -D
server starting
thedoctor@tardis:/pg_data/9.4$ LOG:  database system was shut down
    in recovery at 2015-11-26 07:18:11 GMT
LOG:  entering standby mode
LOG:  restored log file "000000010000000000000007" from archive
LOG:  redo starts at 0/7000060
LOG:  consistent recovery state reached at 0/7000138
LOG:  database system is ready to accept read only connections
cp: cannot stat   /pg_archive/000000010000000000000008   : No such
     file or directory
LOG:  started streaming WAL from primary at 0/8000000 on timeline 1
```

And now for something completely different

Check the slave is in recovery

```
psql -p 5433 postgres
psql (9.4.5)
Type "help" for help.

postgres=# SELECT pg_is_in_recovery();
 pg_is_in_recovery
-------------------
 t
(1 row)
```

And now for something completely different

Troubleshooting

- ERROR: requested WAL segment XXXX has already been removed
- Archive/pg_xlog filling up on the slave
- Slave crashes because of invalid pages
- High lag between the master and the slave

Questions?

Questions?

Contacts and license

- Twitter: 4thdoctor_scarf
- Blog:http://www.pgdba.co.uk
- Brighton PostgreSQL Meetup: http://www.meetup.com/Brighton-PostgreSQL-Meetup/

Credits

- Montparnasse derailment: Source Wikipedia, Public Domain, credited to the firm Levy & fils
- Flail picture: Copyright Tim Avatar Bartel - The flail belongs to his girlfriend
- The two doctors: Copyright Federico Campoli
- The phantom's playground: Copyright Federico Campoli
- The pedestrian seagull: Copyright Federico Campoli

PostgreSQL 9.4 - streaming replication

Federico Campoli

27 Nov 2015