Backup and recovery with PostgreSQL

Mostly harmless

Federico Campoli

Brighton PostgreSQL Users Group

13 June 2016

Table of contents

1. Like tears in the... ACID
2. Crash, Recovery and their band of merry men
3. Point in Time and Recovery in space
4. A heap of broken WALs
5. And now for something completely different
6. Wrap up

Table of contents

1. Like tears in the... ACID
2. Crash, Recovery and their band of merry men
3. Point in Time and Recovery in space
4. A heap of broken WALs
5. And now for something completely different
6. Wrap up

Like tears in the... ACID

Wikipedia, By Source, Fair use

Failure is not an option...

- The hardware is subject to faults.
- Losing the storage makes the data infrastructure inaccessible, maybe for good.
- Human errors, like not filtered delete or table drop can destroy your data.
- A solid backup strategy is the best protection when the disaster strikes.

Logical vs physical

Backup can be implemented in different ways. We'll check the two popular options available in PostgreSQL.

- The logical backup with pg_dump
- The physical backup with the PITR or standby servers

pg_dump at glance

pg_dump is the PostgreSQL's utility for saving consistent backups

- Supports local or remote connections
- Doesn't affects the normal database activity
- It blocks the DML for the relations backed up
- Supports the data dump in multiple jobs (9.3+)
- Can save partial backups (Warning!)

Under the hood

When started for a full backup pg_dump queries the PostgreSQL's system catalogue. In particular

- Starts a new read only transaction
- If launched in parallel jobs exports the snapshot
- Builds the DML from saved relations
- Dumps the DML and the data

The backup formats

When specified the switch -F is possible to save the data in several formats

- p Plain format
- c Custom format
- d Directory format

The plain format

- Outputs a plain SQL script
- Default output to stdout!!!!
- No compression
- Suitable for direct load using a client like psql
- Not flexible at restore time
- md5 checksum simple

The custom format

- Outputs in binary file
- Compression
- Requires pg_restore to load the data into a database
- Supports parallel restore only
- Very flexible at restore time
- md5 checksum simple

The directory format

- Outputs in a directory with a binary toc file
- The table's data is saved in gzipped files
- Compression
- Requires pg_restore to load the data into a database
- Supports parallel backup and restore
- Very flexible at restore time
- md5 checksum can be tricky

Table of contents

1 Like tears in the... ACID

2 Crash, Recovery and their band of merry men

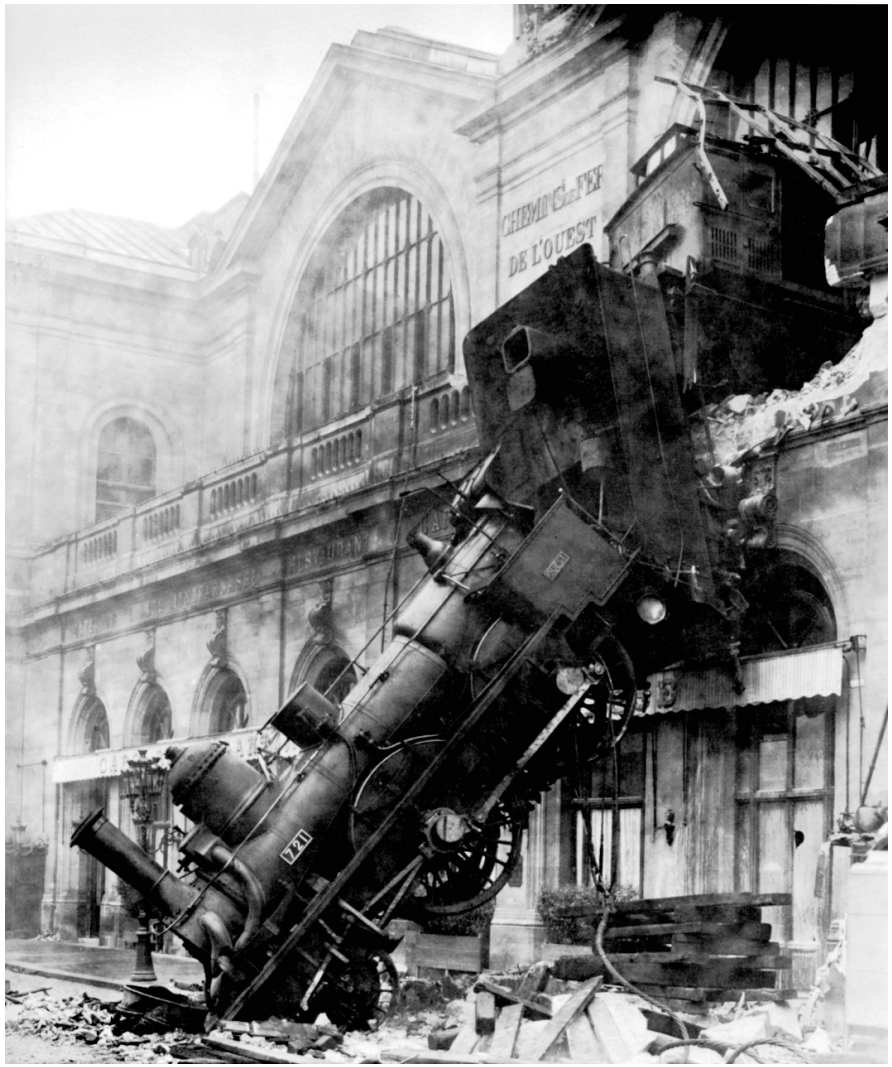
3 Point in Time and Recovery in space

4 A heap of broken WALs

5 And now for something completely different

6 Wrap up

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.

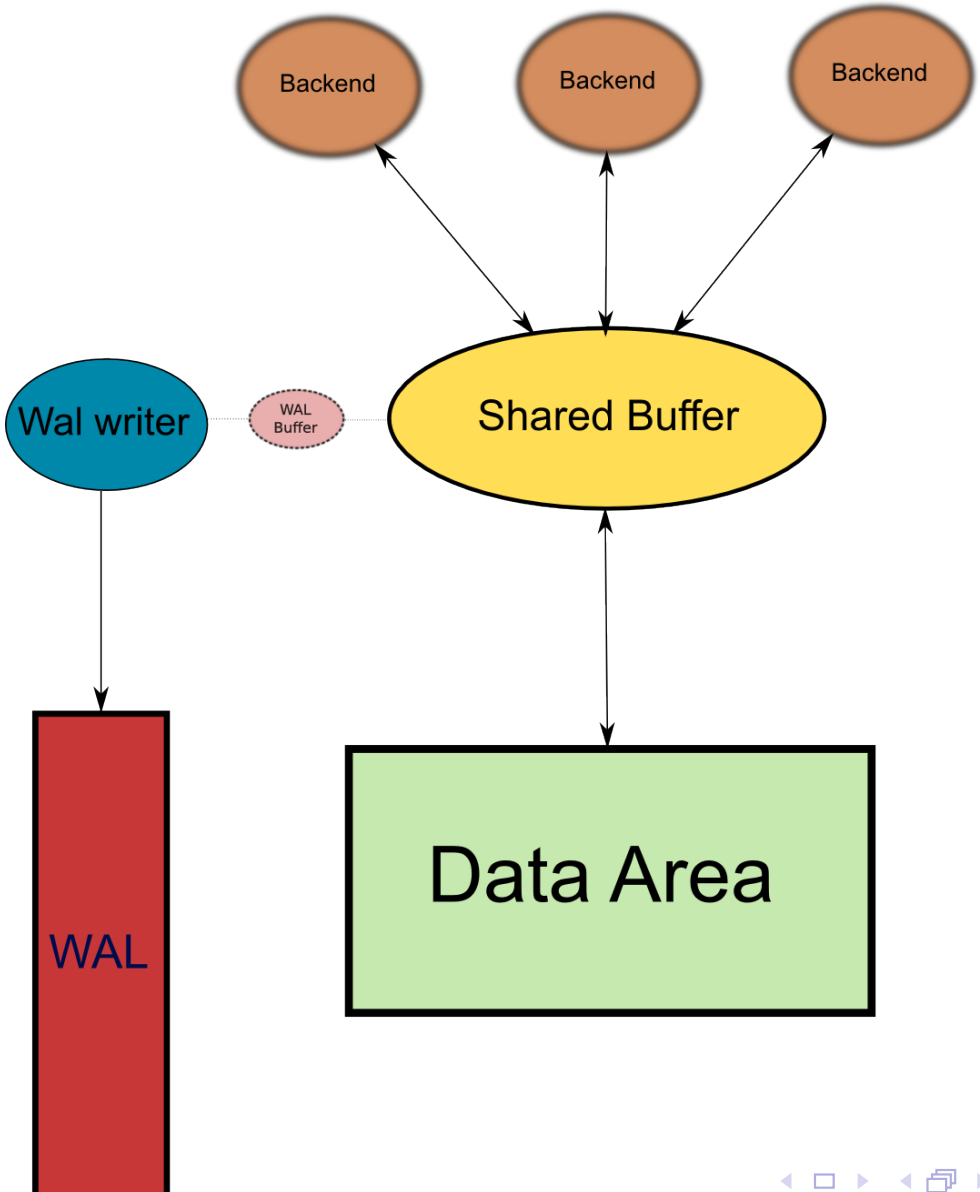
Source Wikipedia

Crash, Recovery and their band of merry men

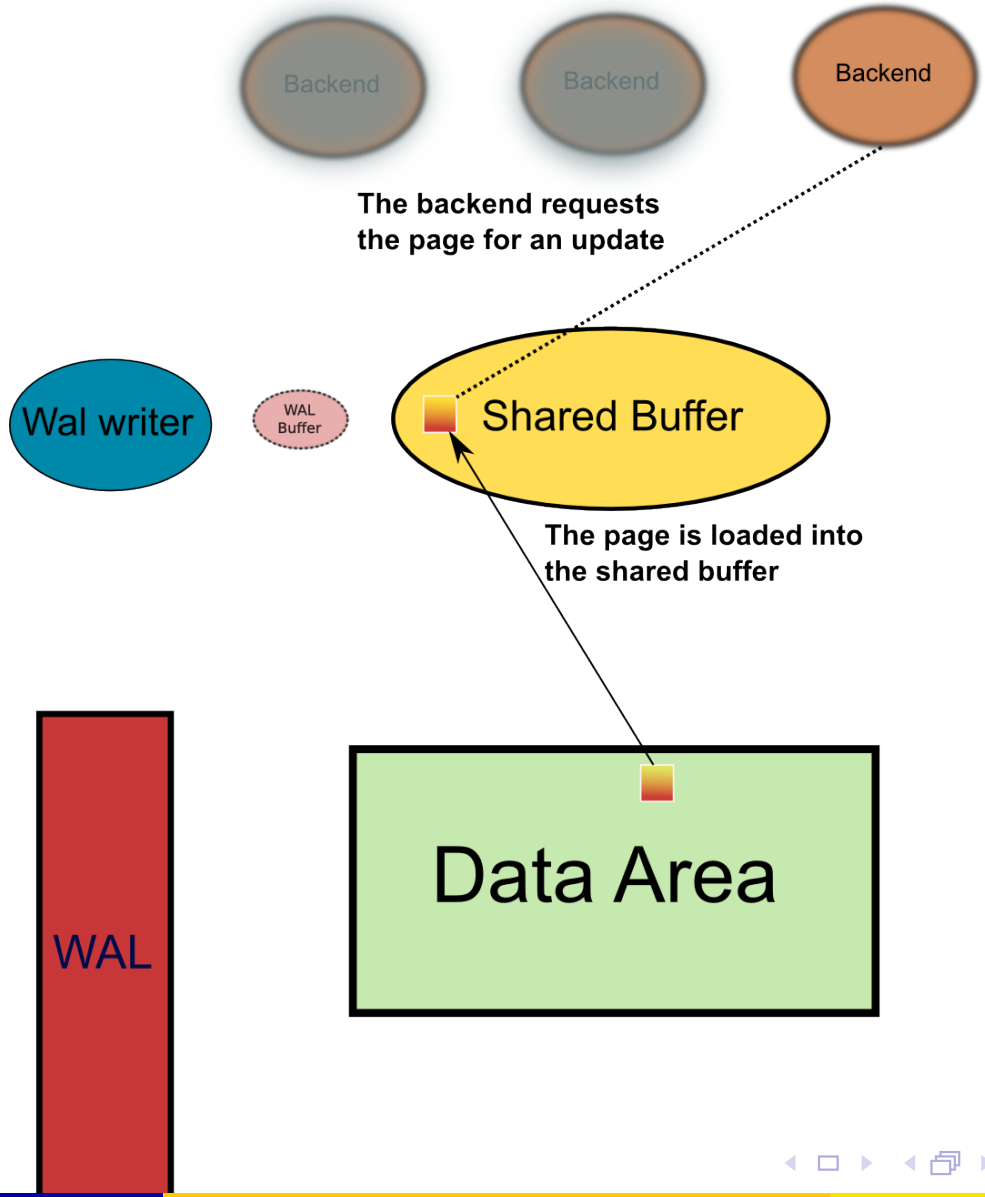
PostgreSQL implements the durability using the Write Ahead Logging.

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.

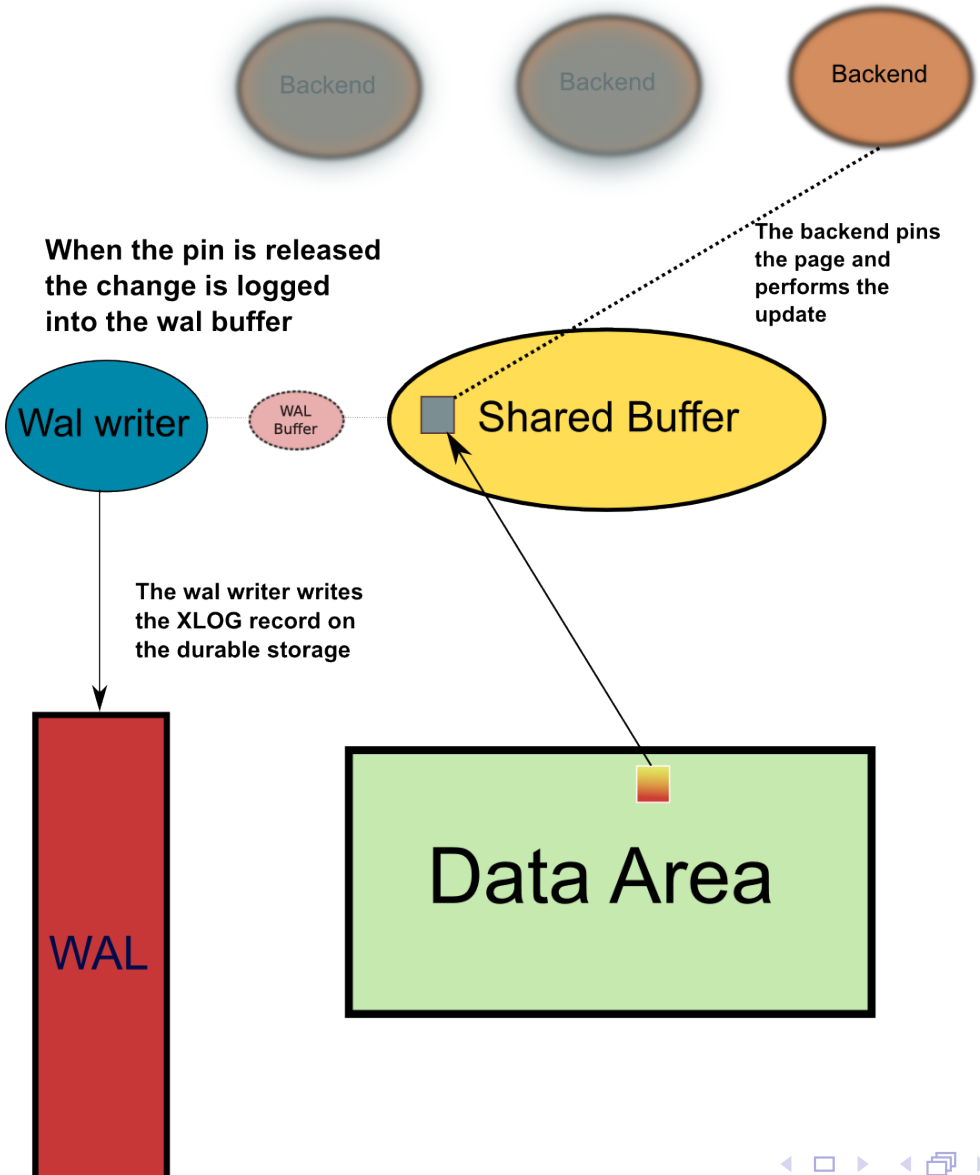
Crash, Recovery and their band of merry men



Crash, Recovery and their band of merry men



Crash, Recovery and their band of merry men



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

1 Like tears in the... ACID

2 Crash, Recovery and their band of merry men

3 Point in Time and Recovery in space

4 A heap of broken WALs

5 And now for something completely different

6 Wrap up

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

1 Like tears in the... ACID

2 Crash, Recovery and their band of merry men

3 Point in Time and Recovery in space

4 A heap of broken WALs

5 And now for something completely different

6 Wrap up

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

1 Like tears in the... ACID

2 Crash, Recovery and their band of merry men

3 Point in Time and Recovery in space

4 A heap of broken WALs

5 And now for something completely different

6 Wrap up

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.

Table of contents

1 Like tears in the... ACID

2 Crash, Recovery and their band of merry men

3 Point in Time and Recovery in space

4 A heap of broken WALs

5 And now for something completely different

6 Wrap up

pg_dump in a nutshell

- Saves statical snapshots of the database
- It's logical
- The restore can take long time
- Doesn't saves the data when the backup in progress
- It's very mature and reliable
- Disaster recovery

PITR in a nutshell

- Saves an on going backup
- It's physical
- The restore can be very fast
- Saves the entire cluster
- It's reliable
- Disaster recovery

Standby in a nutshell

- The server is up and running replaying the changes
- It's physical
- The recovery is almost immediate
- Saves the entire cluster
- It's reliable
- Can be used for load balancing
- High availability

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

Backup and recovery with PostgreSQL

Mostly harmless

Federico Campoli

Brighton PostgreSQL Users Group

13 June 2016