Point-in-time Recovery, target 2020

PgBE meetup

Stefan Fercot

5 March 2020

DALIBO
L'expertise PostgreSQL

Who Am I?

- Stefan Fercot
- aka. pgstef
- https://pgstef.github.io
- PostgreSQL user since 2010
- pgBackRest fan
- @dalibo since 2017

Dalibo

- Services

Support Training Advice

- Based in France
- Contributing to PostgreSQL community

DALIBO
L'expertise PostgreSQL

Introduction

- What is WAL?
- Point-In-Time Recovery (PITR)
  - WAL archives
  - File-system-level backup
  - Restore
- PITR Tools

L'expertise PostgreSQL

What is WAL?

- write-ahead log
  - transaction log (aka xlog)
- usually 16 MB (default)
  - `--wal-segsize` *initdb* parameter to change it
- pg_xlog (<= v9.6) -> pg_wal (v10+)
- designed to prevent data loss in most situations

Write-Ahead Log (WAL)

- transactions written sequentially
  - COMMIT when data are flushed to disk
- WAL replay after a crash
  - make the database consistent

Data modifications

- transactions modify data in `shared_buffers`
- checkpoints and background writer…
  - … push all dirty buffers to the storage

Point-In-Time Recovery (PITR)

- combine
  - file-system-level backup
  - continuous archiving of WAL files
- restore the file-system-level backup and replay archived WAL files

Benefits

- live backup
- less data-losses
- not mandatory to replay WAL entries all the way to the end

Drawbacks

- complete cluster backup…
  - … and restore
- big storage space (data + WAL archives)
- WAL clean-up blocked if archiving fails
- not as simple as `pg_dump`

WAL archives

- 2 possibilities
  - archiver process
  - `pg_receivewal` (via *Streaming Replication*)

Archiver process

- configuration (`postgresql.conf`)
  - `wal_level = replica`
  - `archive_mode = on` or `always`
  - `archive_command = '... some command ...'`
  - `archive_timeout = 0`
- don’t forget to flush the file on disk!

DALIBO
L'expertise PostgreSQL

`pg_receivewal`

- archiving via *Streaming Replication*
- writes locally WAL files
- supposed to get data faster than the archiver process
- replication slot advised!

Benefits and drawbacks

- archiver process
  - easy to setup
  - maximum 1 WAL possible to lose
- `pg_receivewal`
  - more complex implementation
  - only the last transactions are lost

File-system-level backup

- `pg_basebackup`
- manual steps

`pg_basebackup`

- takes a file-system-level copy
  - using *Streaming Replication* connection(s)
- collects WAL archives during (or after) the copy
- no incremental backup

```
$ pg_basebackup --format=tar --wal-method=stream \
 --checkpoint=fast --progress -h HOSTNAME -U NAME \
 -D DIRECTORY
```

Manual steps

- `pg_start_backup()`
- manual file-system-level copy
- `pg_stop_backup()`

`pg_start_backup()`

`SELECT pg_start_backup (`

- `label` : arbitrary user-defined text
- `fast` : immediate checkpoint?
- `exclusive` : exclusive mode?

`)`

Exclusive mode

- easy to use but deprecated since 9.6
- `pg_start_backup()`
  - writes `backup_label`, `tablespace_map`
- works only on primary servers

Non-exclusive mode

- `pg_stop_backup()`
  - executed in the same connection as `pg_start_backup()`!
  - returns `backup_label` and `tablespace_map` content

Data copy

- copy data files while PostgreSQL is running
  - *PGDATA* directory
  - tablespaces
- inconsistency protection with WAL archives
- ignore
  - `postmaster.pid`, `postmaster.opts`, `pg_internal.init`
  - `log`, `pg_wal`, `pg_replslot`,…
- don’t forget configuration files!

`pg_stop_backup()`

`SELECT * FROM pg_stop_backup (`

- exclusive
- wait_for_archive

`)`

- on primary server
  - automatic switch to the next WAL segment
- on standby server
  - consider using `pg_switch_wal()` on the primary…

Restore

- recovery procedure is simple but…
  - must be followed carefully!

Recovery steps (1/5)

- stop the server if it’s running
- keep a temporary copy of your PGDATA / tablespaces
  - or at least the `pg_wal` directory
- remove the content of PGDATA / tablespaces directories

Recovery steps (2/5)

- restore database files from your file system backup
  - pay attention to ownership and permissions
  - verify tablespaces symbolic links
- remove content of `pg_wal` (if not already the case)
- copy unarchived WAL segment files

Recovery steps (3/5)

- configure the recovery…
  - before v12: `recovery.conf`
  - after: `postgresql.conf` + `recovery.signal`
- `restore_command = '... some command ...'`
- prevent ordinary connections in `pg_hba.conf` if needed

PostgreSQL 12

Integrate recovery.conf into postgresql.conf

```
recovery.conf settings are now set in postgresql.conf (or other GUC
sources). Currently, all the affected settings are PGC_POSTMASTER;
this could be refined in the future case by case.

Recovery is now initiated by a file recovery.signal. Standby mode is
initiated by a file standby.signal. The standby_mode setting is
gone. If a recovery.conf file is found, an error is issued.

...

pg_basebackup -R now appends settings to postgresql.auto.conf and
creates a standby.signal file.
```

Recovery steps (4/5)

- recovery target:
  - `recovery_target_name`, `recovery_target_time`
  - `recovery_target_xid`, `recovery_target_lsn`
  - `recovery_target_inclusive`
- timeline to follow:
  - `recovery_target_timeline`
- action once recovery target is reached?
  - `recovery_target_action`
  - `pg_wal_replay_resume`

Recovery steps (5/5)

- start the server
- watch the restore process
  - until consistent recovery state reached
- inspect your data

LSN

- log sequence number
  - position of the record in WAL file
  - provides uniqueness for each WAL record

```
=# SELECT pg_current_wal_lsn();
 pg_current_wal_lsn
--------------------
 2/3002020
(1 row)

=# SELECT pg_walfile_name(pg_current_wal_lsn());
     pg_walfile_name
--------------------------
 000000010000000200000003
(1 row)
```

Timelines

- archive recovery complete -> new timeline
  - part of WAL segment file names
  - to identify the series of WAL records generated after that recover
  - `.history` files
- `recovery_target_timeline`
  - default: `latest` (v12+) or `current` (< v12)

WAL filename

- 000000010000000200000003
  - 00000001 : timeline
  - 00000002 : wal
  - 00000003 : segment
- hexadecimal
  - 000000010000000000000001
  - 0000000100000000000000FF
  - 000000010000000100000000
  - …

PITR Tools

- tools make life easier
  - pgBackRest
  - pitrery
  - Barman
  - WAL-G
- providing
  - backup, restore, purge methods
  - archiving commands

pgBackRest

- written in C (since version 2.21)
- custom protocol
  - local or remote operation (via SSH)
- full/differential/incremental backup
- parallel, asynchronous WAL push and get
- Amazon S3 support

pitrery

- set of Bash scripts
  - `archive_wal`
  - `pitrery`
  - `restore_wal`
- *push* mode (*SSH*)
- mono-server
- *tar* or *rsync* backup method

Barman

- written in Python
- remote backups (*pull* mode)
  - via *SSH*
  - or *Streaming Replication*
- handles multiple servers
- `pg_receivewal` & `pg_basebackup` support

WAL-G

- written in Go
- based on WAL-E
- storage
  - Amazon S3
  - Google Cloud
  - Azure
  - local

What is a good database backup tool?

- usable
  - documentation & support
  - out-of-box automatization of various routines
- scalable
  - parallel execution
  - compression
  - incremental & differential backups
- reliable
  - Schrödinger’s backup law
    - *The condition of any backup is unknown until a restore is attempted*

WAL archives

	archive_command	restore_command	`pg_receivewal`
pgBackRest	YES (+ archive-async)	YES (+ archive-async)	NO
pitrery	YES	YES	NO
Barman	YES	YES	YES
WAL-G	YES	YES (+ wal prefetch)	NO

Encryption

		method
pgBackRest	YES	aes-256-cbc
pitrery	NO	
Barman	NO	
WAL-G	YES	S3 server-side / libsodium

Parallel execution

	backup, restore	archiving	parameters
pgBackRest	YES	YES	process-max
pitrery	NO	NO	
Barman	YES rsync	NO	parallel_jobs
WAL-G	YES	YES	WALG_*_CONCURRENCY

Compression

	backups	archives	how?
pgBackRest	YES	YES	gzip
pitrery	YES tar	YES	gzip, pigz, bzip2,…
Barman	NO	YES	gzip, pigz, bzip2,…
WAL-G	YES	YES	lz4, lzma, brotli

Network

	network compression	bandwidth limit
pgBackRest	YES	NO
pitrery	NO	YES rsync
Barman	YES rsync	YES rsync
WAL-G	NO	YES

Incremental backups

		how?

pgBackRest	YES	`--type=incr`
		`--type=diff`

pitrery	YES rsync	hardlinks
Barman	YES rsync	hardlinks
WAL-G	YES	WALG_DELTA_MAX_STEPS WALG_DELTA_ORIGIN

Useful resources

- Devrim Gündüz - WAL: Everything You Want to Know
- PostgreSQL docs - WAL introduction
- PostgreSQL docs - Continuous Archiving and PITR
- Anastasia Lubennikova - Advanced backup methods

DALIBO
L'expertise PostgreSQL

Conclusion

- PITR is
  - reliable
  - fast[er than `pg_dump`]
  - continuous
- tools make life easier
  - choose wisely…

Questions?

DALIBO
L'expertise PostgreSQL