Efficiently Backing up Terabytes of Data with pgBackRest

PGConf US 2016

David Steele

April 20, 2016

Agenda

Why Backup?

- Hardware Failure
  - No amount of redundancy can prevent it
- Replication
  - WAL archive for when async streaming gets behind
  - Sync replica from backup instead of master
- Corruption
  - Can be caused by hardware or software
  - Detection is of course a challenge

Why Backup?

- Accidents
  - So you dropped a table?
  - Deleted your most important account?
- Development
  - No more realistic data than production!
  - May not be practical due to size / privacy issues
- Reporting
  - Use backups to standup an independent reporting server
  - Recover important data that was removed on purpose

Schrödingers Backup

The state of any backup is unknown until a restore is attempted.

Making Backups Useful

- Find a way to use your backups
  - Syncing / New Replicas
  - Offline reporting
  - Offline data archiving
  - Development
- Unused code paths will not work when you need them unless they are tested
  - Regularly scheduled automated failover using backups to restore the old primary
  - Regularly scheduled disaster recovery (during a maintenance window if possible) to test restore techniques

How to Backup?

- pg_dump
- pg_basebackup
- Manual
- Third Party
  - OmniPITR
  - Barman
  - WAL-E
- pgBackRest!

Design

- Rsync powers many database backup solutions but it has some serious limitations
  - Single-threaded
  - One second timestamp resolution
  - Incremental backups require previous backup to be uncompressed
- pgBackRest does not use rsync, tar or any other tools of that type
  - Protocol supports local/remote operation
  - Solves timestamp resolution issue

Multithreaded Backup & Restore

Compression is usually the bottleneck during backup operations but, even with now ubiquitous multi-core servers, most database backup solutions are still single-threaded. pgBackRest solves the compression bottleneck with multithreading.

Utilizing multiple cores for compression makes it possible to achieve 1TB/hr raw throughput even on a 1Gb/s link. More cores and a larger pipe lead to even higher throughput.

Local or Remote Operation

A custom protocol allows pgBackRest to backup, restore, and archive locally or remotely via SSH with minimal configuration. An interface to query PostgreSQL is also provided via the protocol layer so that remote access to PostgreSQL is never required, which enhances security.

Full, Incremental, & Differential Backups

Full, differential, and incremental backups are supported. pgBackRest is not susceptible to the time resolution issues of rsync, making differential and incremental backups completely safe.

Backup Rotation & Archive Expiration

Retention polices can be set for full and differential backups to create coverage for any timeframe. WAL archive can be maintained for all backups or strictly for the most recent backups. In the latter case WAL required to make older backups consistent will be maintained in the archive.

Backup Integrity

Checksums are calculated for every file in the backup and rechecked during a restore. After a backup finishes copying files, it waits until every WAL segment required to make the backup consistent reaches the repository.

Backups in the repository are stored in the same format as a standard PostgreSQL cluster (including tablespaces). If compression is disabled and hard links are enabled it is possible to snapshot a backup in the repository and bring up a PostgreSQL cluster directly on the snapshot. This is advantageous for terabyte-scale databases that are time consuming to restore in the traditional way.

All operations utilize file and directory level fsync to ensure durability.

Backup Resume

An aborted backup can be resumed from the point where it was stopped. Files that were already copied are compared with the checksums in the manifest to ensure integrity. Since this operation can take place entirely on the backup server, it reduces load on the database server and saves time since checksum calculation is faster than compressing and retransmitting data.

Streaming Compression & Checksums

Compression and checksum calculations are performed in stream while files are being copied to the repository, whether the repository is located locally or remotely.

If the repository is on a backup server, compression is performed on the database server and files are transmitted in a compressed format and simply stored on the backup server. When compression is disabled a lower level of compression is utilized to make efficient use of available bandwidth while keeping CPU cost to a minimum.

Delta Restore

The manifest contains checksums for every file in the backup so that during a restore it is possible to use these checksums to speed processing enormously. On a delta restore any files not present in the backup are first removed and then checksums are taken for the remaining files. Files that match the backup are left in place and the rest of the files are restored as usual. Since this process is multithreaded, it can lead to a dramatic reduction in restore times.

Advanced Archiving

Dedicated commands are included for both pushing WAL to the archive and retrieving WAL from the archive.

The push command automatically detects WAL segments that are pushed multiple times and de-duplicates when the segment is identical, otherwise an error is raised. The push and get commands both ensure that the database and repository match by comparing PostgreSQL versions and system identifiers. This precludes the possibility of misconfiguring the WAL archive location.

Asynchronous archiving allows compression and transfer to be offloaded to another process which maintains a continuous connection to the remote server, improving throughput significantly. This can be a critical feature for databases with extremely high write volume.

Tablespace & Link Support

Tablespaces are fully supported and on restore tablespaces can be remapped to any location. It is also possible to remap all tablespaces to one location with a single command which is useful for development restores.

File and directory links are supported for any file or directory in the PostgreSQL cluster. When restoring it is possible to restore all links to their original locations, remap some or all links, or restore some or all links as normal files or directories within the cluster directory.

Compatibility with PostgreSQL $\geqslant$ 8.3

pgBackRest includes support for versions down to 8.3, since older versions of PostgreSQL are still regularly utilized.

Performance

Parameters	pgBackRest	rsync
threads: 1 network compression: l3 destination compression: none	141 Seconds	124 Seconds (.13X Faster)
threads: 2 network compression: l3 destination compression: none	84 Seconds (1.48X Faster)	N/A
threads: 1 network compression: l6 destination compression: l6	334 Seconds (1.52X Faster)	510 Seconds
threads: 2 network compression: l6 destination compression: l6	174 Seconds (2.93X Faster)	N/A

Demonstration

Live Demo — this should be fun!

Questions?

website: `http://www.pgbackrest.org`

email: david@pgbackrest.org
email: david@crunchydata.com

releases: `https://github.com/pgbackrest/pgbackrest/releases`

slides & demo: `https://github.com/dwsteele/conference/releases`