PostgreSQL Backup Strategies

PostgreSQL Conference Europe 2012
Prague, The Czech Republic

Magnus Hagander
magnus@hagander.net

Replication!

- But I have replication!
- To multiple nodes!
- It's even in *the cloud*!

What about clustering?

- Yeah, pretty much the same

But my SAN is 100% up!

- Really?

But my SAN is 100% up!

- Really?
- No, *really?!*

Backup planning

- Backup interval
- Backup retention
- Performance impacts

Restore planning

- Time spent taking backups usually not important
- Time it takes to restore is *critical*
- Consider multi-stage solutions

PostgreSQL options

- Logical backups
  - pg_dump
- Physical backups
  - Filesystem snapshots
  - pg_basebackup
  - “Manual” base backups

Logical backups

- SQL script dump of schema + data
- Restored through SQL commands
- Great flexibility
- Not the greatest for performance

pg_dump

- This is your main tool
  - Dumps a single database
- Regular PostgreSQL connection
- Guarantees consistent snapshot across database
- Single threaded
  - (for now..)

pg_dump

- Supports multiple output formats
  - *Always* use “custom” format (-Fc)
  - Compressed by default (-Fc)
- Supports dumping separate objects
  - For backups, always dump whole database

pg_dump system impact

- Runs regular COPY queries
- Uses single backend
- Does not ruin PostgreSQL cache
  - “ring buffer” strategy used
- Can potentially ruin filesystem cache
- Writing of dump file causes I/O

pg_dump compression

- Compression happens in *pg_dump*
- Can be used for throttling
  - Typical “breakpoint” at 3-5
  - Higher becomes CPU bound
  - Lower becomes I/O bound

pg_dump ssh tunnel

-
  ```
  ssh dbserver "pg_dump -Z9 -Fc -U postgres mydb" > mydb.dump
  ```

pg_dump ssh tunnel

-
  ```
  ssh dbserver "pg_dump -Z9 -Fc -U postgres mydb" > mydb.dump
  ```

-
  ```
  ssh -o "Compression=no" magh.u.bitbit.net "pg_dump -Z9 -Fc -U postgres mydb" > mydb.dump
  ```

Restoring from pg_dump

- Use pg_restore
  - Reads “custom” format dumps
  - Regular connection
- Full database restore
  - “Recover from backups”
- Partial database restore
  - “Create staging env”
  - “Single table restore”

Restore performance

- Regular COPY
  - Followed by CREATE INDEX
  - And ADD CONSTRAINT
- *Very slow for large databases!*

Restore performance

- Use -1 flag
- Full restore as single transaction
- Enables multiple optimizations
  - Particularly if WAL archiving not enabled
- Empty database in case of crash

Restore performance

- Restore in parallel sessions
  - -j <n>
- Each object still in one session
- *Not* compatible with -1
  - Need to pick one
  - -j usually faster

Restore performance

- Turn fsync=off
  - Last resort
  - But quite useful
- Don't forget to turn it back on!
  - (Yes, it happens)
- Don't forget to flush OS caches!
  - (Yes, you'll get corruption)

Don't forget!

- pg_dumpall -g
  - Users, groups, tablespaces

Physical backups

Physical backups

- PostgreSQL stores database in files
- We can backup those files...
- No need to parse or query
  - Thus faster!
- Architecture, version, compile flags and paths must be identical
- Only full cluster backups

Offline backups

- Easiest possible way
  - Stop PostgreSQL, take backup, start PostgreSQL
- Backup files any way possible
  - Tar, copy, filesystem snapshot etc
- Not to be ignored...

Simple snapshot backups

- Filesystem/SAN snapshots while database is running
- Requires atomic snapshot across all tablespaces
  - Including pg_xlog
- Mainly useful in small installations

Online base backups

- Non-constrained filesystem level backups
- Recoverable in combination with transaction log
- With or without log archive
- Provides base for PITR

Online base backups

- Integrated base backups
  - On top of replication protocol
- Enable replication!
  - wal_level=archive
  - max_wal_senders=2

Online base backups

-
  ```
  pg_basebackup
     -U postgres
     -D backup
     -P
     -x
  ```

- Requires “enough” WAL to stay around
- Generates complete data directory

Log archiving

- As log is generated, send to archive
- On restoring, fetch back from archive
  - Start from base backup
  - "Roll forward" through archived log
  - Stop at any point

Log archiving in PostgreSQL

- `archive_mode=on`
  - Starts the log archiver
- `archive_command=<something>`
  - “take file x and store it under the name y”
- `restore_command=<something>`
  - “give me back the file you stored under name y”

Log archiving limitations

- Always 16Mb segments
  - `archive_timeout=<n>`
- Too much or not enough
- Replication solves problem in 9.1
- 9.2: pg_receivexlog

Base backups for PITR

- pg_basebackup without -x
- Manual method:
  - SELECT pg_start_backup();
  - <copy files>
    - Copy files, e.g. cp/tar
    - Rsync
    - SAN snapshots
  - SELECT pg_stop_backup();

pg_basebackup system impact

- Reads all data, generates lots of I/O
- pg_basebackup single threaded
  - This is probablyusually a good thing
- Sequential reads
- (Optional) compression happens in pg_basebackup, not server

Restore performance

- Depends on “distance to base backup”
- Read back all log files, replays
  - Generates random writes
  - Single threaded as well
    - (more or less)
- Multiple generations of base backups

Backup strategies

Please make backups

How to back up

- You definitely want online physical backups
- You almost certainly want PITR
- You probably want pg_dump
  - If you can afford it

Backup retention

- Comes back to business requirements
- How far back does it *make sense* to restore data?
- And at what resolutions?

Log file/base backup

- Restore requires base backup + all log files since with no “holes”
- Keep fewer base backups but all logs
- Keep fewer logs but more base backups

Backup vs replication

- You probably want both
- Backups are more important
- Replication good for hardware failure
- And allows for *much* shorter service interruption

Lagged behind replicas

- Using file based replication
- Introduce delay in the system
  - E.g. 1 hour or 12 hours
- Roll forward replica instead of restoring from backups

Testing your backups

Testing your backups

- We all know we should
- And we seldom do

Use for staging and dev

- Restore from backup instead of deploy from master
- *Do not automate!*

Thank you!

Questions?
Share your stories!

Twitter: @magnushagander
http://blog.hagander.net/
magnus@hagander.net