The Future of PostgreSQL
High Availability

Robert Hodges - Continuent, Inc.
Simon Riggs - 2ndQuadrant
Agenda

/ Introductions
/ Framing the High Availability (HA) Problem
/ Hot Standby + Log Streaming
/ The PostgreSQL HA Manifesto
/ Questions
About Us

Simon Riggs -- Key PostgreSQL HA Contributor
- PITR, pg_standby, hot standby, etc.

Robert Hodges -- Architect of Tungsten Clustering
- Tungsten Replicator for MySQL & PostgreSQL, backups, distributed management, etc.

Continuent: Cross-platform database clustering
- Protect data
- Improve availability
- Scale performance

2ndQuadrant: PostgreSQL services and core dev
- Services
- Education
- Support
Framing the Problem: Database High Availability
DBMS High Availability Made Simple

**Availability**: Degree to which a system is up and running.

**Keys to High Availability**

1. Minimize failures
2. Keep downtime including repairs as short as possible
3. Don’t lose more data than you absolutely have to
What Are Key Causes of Downtime?

/ **Crashes** -- Hardware or software component fails

/ **Scheduled maintenance** – Upgrade/service components

/ **Migration** — Moving between DBMS versions and hardware architectures

/ **Administrative errors** — Accidents that delete data or cause components not to work

Thought exercise: which accounts for the most down-time?
Who Needs High Availability?

/ Small/medium business applications
  • Idiot-proof installation and management

/ Embedded medical data processing
  • Unattended operation
  • Never lose a transaction

/ Hosted website intrusion reporting
  • Burst updates to 100K INSERTs per second
  • Massive data volumes

/ Hosted CRM (Customer Relationship Management)
  • Fail-back options for system upgrades
  • Creation of reporting databases

/ Internet Service Provider
  • Shared DBMS instances
  • Transparent migration of users between instances
Shared vs. Redundant Resources

- Shared resources create single points of failover (SPOFs)
- More redundancy == higher availability

Redundant data approach covers more use cases with less capable hardware

Shared disk approach requires internal redundancy; limited data protection and fewer use cases
Backups and Point-In-Time-Recovery

- Backups are first line of defense for availability
- **Point-in-time-recovery** restores database state to a particular:
  - Point in calendar time, or
  - Transaction ID
- **Provisioning** copies directly from one database instance to another
Physical vs. Logical Replication

- Databases can update either at disk or logical level, hence two replication approaches
- Log records -- Databases apply them automatically during recovery
- SQL statements -- Clients send SQL to make changes

**Physical Replication**
Replicate log records/events to create bit-for-bit copy

Transparent, high performance, hard to cross architectures and versions, limitations on updates

**Logical Replication**
Replicate SQL to create equivalent data

Flexible, fewer/different restrictions, allow schema differences, replicas allow reads
Asynchronous vs. Synchronous

Replicating is like buying a car--there are lots of ways to pay for it:

- $0 down - Pay later; hope nothing goes wrong
- Down payment - Pay some so less goes wrong later
- Cash - Pay up front and it’s yours forever

<table>
<thead>
<tr>
<th>Asynchronous Replication</th>
<th>Semi-Synchronous Replication</th>
<th>Synchronous Replication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commit now, replicate later</td>
<td>Replicate to at least one other database</td>
<td>Replicate fully to all other databases</td>
</tr>
</tbody>
</table>

Lose data but robust against network failure

Trade-off data loss vs. partition handling

Network fails --> you stop

PG Day EU 2009 - Paris
Simple vs. Complex

Simple systems are almost always more available
Complexity is the #1 enemy of high availability

Built-in database replication creates simple system with few/no additional ways to fail
Complex combinations are hard to understand, test, and manage; create new failure modes
Hot Standby and Log Streaming
PostgreSQL 8.4 Warm Standby

Master

- PostgreSQL
- pg_xlogs Directory
- WAL Files

Standby

- PostgreSQL
- pg_xlogs Directory
- WAL Files
- Archive Directory
- Archived WAL Files

rsync to standby

Continuous recovery

pg_standby
Advantages of Warm Standby

- Simple
- Completely transparent to applications
- Very low performance overhead
  - E.g. no extra writes from triggers
- Supports point-in-time recovery
- Works over WAN as well as LAN
- Has reasonable recovery of master using rsync
- Very reliable solution -- if recovery works warm standby works
- Requires careful management to use effectively
Limitations of Warm Standby

1. **Utilization -- Cannot open the standby**
   - To bring up the standby for queries you must end recovery
   - Standby hardware is idle
   - Difficult to track state of recovery since you cannot query log position

2. **Data Loss -- Warm standby transfers only full WAL files**
   - Can bound loss using `archive_timeout`
   - Low values create large numbers of WAL files; complicate point-in-time recovery
   - Workarounds using DRBD, etc. are complex
Introducing Hot Standby

/ Allows users to connect to standby in read-only mode
  • Allowed: SELECT, SET, LOAD, COMMIT/ROLLBACK
  • Disallowed: INSERT, UPDATE, DELETE, CREATE, 2PC, SELECT ... FOR SHARE/UPDATE, nextval(), LISTEN, LOCK,
  • No admin commands: ANALYZE, VACUUM, REINDEX, GRANT

/ Simple configuration through recovery.conf
# Hot standby
recovery_connections = 'on'

/ Performance Overhead
  • Master: < 0.1% impact from additional WAL
  • Standby: 2% CPU impact, but we're I/O bound anyway

/ Can come out of recovery while queries are running
Hot Standby Query Conflicts

/ Master: Connections can interfere and deadlock

/ Standby: Queries can conflict with recovery
  • Recovery always wins

/ Causes of conflicts
  • Cleanup records (HOT/VACUUM)
  • Access exclusive locks
  • DROP DATABASE
  • DROP TABLESPACE
  • Very long queries

/ Conflict resolution
  • Wait, then Cancel - Controlled by max_standby_delay
  • Avoid - Dblink
Introducing Log Streaming

Continuous replication to standby

Archiving

Archived WAL Files

Archive Directory
Configuration and Usage

/ Log streaming layers on top of existing warm standby log shipping

/ Configuration through postgresql.conf + recovery.conf

# Recovery.conf log streaming options
standby_mode = 'on'
primary_conninfo = 'host=172.16.238.128
    port=5432  user=postgres'
trigger_file = '/path_to/trigger'

/ Multiple standby servers allowed
/ Failure of one standby does not affect others
/ Management is not simple - must coordinate provisioning & WAL shipping to set up/restart
What Does This Get Us?

• Hot standby enhances utilization
• Hot standby makes standby monitoring very simple
• Hot standby heats up FS cache and shared buffers
• Log streaming reduces the data loss window and shortens failover
• Hot standby + log streaming will be the favored basic availability solution and will largely replace:
  • Master/slave availability using SLONY/Londiste/PG Replicator
  • Disk block replication
  • Shared disk failover

• So are we there yet??
The PostgreSQL HA Manifesto
Developing the PostgreSQL HA Roadmap

/ What can we learn from the neighbors?
/ Four features to round out PostgreSQL HA
MySQL Master Master Replication

- Logical replication is built in -- no triggers
- Covers all SQL including DDL
- Handles maintenance very well (painless resync, application upgrades, cross architecture/version)
Google Semi-Synchronous Replication

- Quorum algorithm -- Commits block until at least one slave responds affirmatively
- Protects data but avoids system freeze if a slave is unavailable
- Released as patch to MySQL; not widely available yet

MySQL DBMS

Commit succeeds when > 0 slaves respond affirmatively
Oracle Data Guard

- Oracle Data Guard moves transaction (redo) logs
- Protection modes include async/sync replication
- Physical standby is bit-for-bit copy, readonly
- Logical standby allows readable, updatable copy
- WAY better than RAC or Streams
Oracle Flash Back

/ Flash Back Query builds PITR into the DBMS
/ Select any SCN (System Commit #) for which logs are available
/ Flash back query to recover deleted data
/ Flash back instance to convert failed master to slave
  • Sounds better than rsync, doesn’t it?
Drizzle Pluggable Replication

- Public replication protocol (Google Proto Buffers)
- Pluggable replication -- enable new replication types
- Sync/async replication
- Support for all SQL operations, not just DML
PostgreSQL HA: Synchronous Replication

Flexible, synchronous replication
• Physical replication is the beginning…

Selectable apply modes
• Submitted to replication
• Received by slave
• Applied by slave

Selectable quorum semantics
• Async
• Semi-sync
• Synchronous

Enables any application that values data to trade off durability vs. availability

Vendor solution jump off: configuration and management
PostgreSQL HA: Real-Time PITR

- Implement Flash Back for PostgreSQL
  - Implementations range from straightforward to very hard
  - Use zoned snapshots to pick points in past where data remain visible to R/O transactions
  - Extra credit: Let PostgreSQL revert to a snapshot

- Usage: Allow users to recover data from specific point in time--like built-in time delay replication. Snapshot reversion simplifies master recovery
  - Vendor jump-off point: Set up and manage snapshots
PostgreSQL HA: VLDB High Availability

/ Multiple simultaneous backups (only one now supported)
  • Backup ref counts to allow more than one customer at a time

/ Incremental backup with WAL synchronization

/ Efficient recovery of large masters after failover

/ Vendor solution jump-off -- Management, fast backup/restore utilities, incremental backup solutions
PostgreSQL HA: Logical Replication

/ Supplement WAL to allow SQL generation
  • Keys
  • Schema definitions
  • Recover DDL statements in “actionable” form (e.g., XML)

/ Extensible replication plug-ins a la Drizzle
  • Intercept data as they are written to log
  • Ability to hold commits to mark transactions (e.g., global IDs) and implement synchronous replication
  • Handle two-phase commit issues
  • Loadable through SQL without weird syntax extensions

/ Provide built-in reference implementation

/ Open source/vendor jump-off: Migration, multi-master, filtering, data consistency checking and repair
Summary and Questions
Summary

- Hot standby + log streaming provide sound built-in “simple” HA
- PostgreSQL HA manifesto = roadmap to a complete solution for high availability with jump-offs to vendor solutions
- Tell us what features you need!
Information/Contact

Continuent Web Site:
http://www.continuent.com

2ndQuadrant Web Site:
http://www.2ndquadrant.com