PostgreSQL replication strategies

Understanding High Availability and choosing the right solution

emmanuel.cecchet@continuent.com
emmanuel.cecchet@epfl.ch

Slides available at http://sequoia.continuent.org/Resources
What Drives Database Replication?

/ **Availability** – Ensure applications remain up and running when there are hardware/software failures as well as during scheduled maintenance on database hosts

/ **Read Scaling** – Distribute queries, reports, and I/O-intensive operations like backup, e.g., on media or forum web sites

/ **Write Scaling** – Distribute updates across multiple databases, for example to support telco message processing or document/web indexing

/ **Super Durable Commit** – Ensure that valuable transactions such as financial or medical data commit to multiple databases to avoid loss

/ **Disaster Recovery** – Maintain data and processing resources in a remote location to ensure business continuity

/ **Geo-cluster** – Allow users in different geographic locations to use a local database for processing with automatic synchronization to other hosts
High availability

/ The magic nines

<table>
<thead>
<tr>
<th>Percent uptime</th>
<th>Downtime/month</th>
<th>Downtime/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.0%</td>
<td>7.2 hours</td>
<td>3.65 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>43.2 minutes</td>
<td>8.76 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>4.32 minutes</td>
<td>52.56 minutes</td>
</tr>
<tr>
<td>99.999%</td>
<td>0.43 minutes</td>
<td>5.26 minutes</td>
</tr>
<tr>
<td>99.9999%</td>
<td>2.6 seconds</td>
<td>31 seconds</td>
</tr>
</tbody>
</table>
Few definitions

/ **MTBF**
  - Mean Time Between Failure
  - Total MTBF of a cluster must combine MTBF of its individual components
  - Consider mean-time-between-system-abort (MTBSA) or mean-time-between-critical-failure (MTBCF)

/ **MTTR**
  - Mean Time To Repair
  - How is the failure detected?
  - How is it notified?
  - Where are the spare parts for hardware?
  - What does your support contract say?
Outline

- Database replication strategies
- PostgreSQL replication solutions
- Building HA solutions
- Management issues in production
Problem: Database is the weakest link

/ Clients connect to the application server
/ Application server builds web pages with data coming from the database
/ Application server clustering solves application server failure
/ Database outage causes overall system outage
Disk replication/clustering

- Eliminates the single point of failure (SPOF) on the disk
- Disk failure does not cause database outage
- Database outage problem still not solved
Database clustering with shared disk

- Multiple database instances share the same disk
- Disk can be replicated to prevent SPOF on disk
- No dynamic load balancing
- Database failure not transparent to users (partial outage)
- Manual failover + manual cleanup needed
Master/slave replication

- Lazy replication at the disk or database level
- No scalability
- Data lost at failure time
- System outage during failover to slave
- Failover requires client reconfiguration
Scaling the database tier
Master-slave replication

/ Pros
  • Good solution for disaster recovery with remote slaves

/ Cons
  • failover time/data loss on master failure
  • read inconsistencies
  • master scalability
Scaling the database tier
Atomic broadcast

/ Pros
  • consistency provided by multi-master replication

/ Cons
  • atomic broadcast scalability
  • no client side load balancing
  • heavy modifications of the database engine
Scaling the database tier – SMP

/ **Pros**
  - Performance

/ **Cons**
  - Scalability limit
  - Limited reliability
  - Cost

---

**Pros**

- Performance

**Cons**

- Scalability limit
- Limited reliability
- Cost

---

Well-known **hardware** + database vendors here

Pros

- Performance

Cons

- Scalability limit
- Limited reliability
- Cost

---

Well-known hardware + database vendors here

---

Web frontend

Internet

App. server

Database

Vendor here
Middleware-based replication

/ Pros
• no client application modification
• database vendor independent
• heterogeneity support
• pluggable replication algorithm
• possible caching

/ Cons
• latency overhead
• might introduce new deadlocks
Transparent failover

- Failures can happen
  - in any component
  - at any time of a request execution
  - in any context (transactional, autocommit)

- Transparent failover
  - masks all failures at any time to the client
  - perform automatic retry and preserves consistency
Outline

/ Database replication strategies
/ PostgreSQL replication solutions
/ Building HA solutions
/ Management issues in production
## PostgreSQL replication solutions compared

<table>
<thead>
<tr>
<th>Feature</th>
<th>pgpool-I</th>
<th>pgpool-II</th>
<th>PGcluster-I</th>
<th>PGcluster-II</th>
<th>Slony-I</th>
<th>Sequoia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replication type</td>
<td>Hot standby</td>
<td>Multi-master</td>
<td>Multi-master</td>
<td>Shared disk</td>
<td>Master/Slave</td>
<td>Multi-master</td>
</tr>
<tr>
<td>Commodity hardware</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Application modifications</td>
<td>No</td>
<td>No</td>
<td>Yes if reading from slaves</td>
<td>No</td>
<td>Yes if reading from slaves</td>
<td>Client driver update</td>
</tr>
<tr>
<td>Database modifications</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PG support</td>
<td>&gt;=7.4 Unix</td>
<td>&gt;=7.4 Unix</td>
<td>7.3.9, 7.4.6, 8.0.1 Unix</td>
<td>8.? Unix only?</td>
<td>&gt;= 7.3.3</td>
<td>All versions</td>
</tr>
<tr>
<td>Data loss on failure</td>
<td>Yes</td>
<td>Yes?</td>
<td>No?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Failover on DB failure</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No if due to disk</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Transparent failover</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Disaster recovery</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No if disk</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Queries load balancing</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## PostgreSQL replication solutions compared

<table>
<thead>
<tr>
<th>Feature</th>
<th>pgpool-I</th>
<th>pgpool-II</th>
<th>PGcluster-I</th>
<th>PGcluster-II</th>
<th>Slony-I</th>
<th>Sequoia</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read scalability</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Write scalability</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Query parallelization</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No?</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Replicas</td>
<td>2</td>
<td>up to 128</td>
<td>LB or replicator limit</td>
<td>SAN limit</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>Super durable commit</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Add node on the fly</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes (slave)</td>
<td>Yes</td>
</tr>
<tr>
<td>Online upgrades</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes (small downtime)</td>
<td>Yes</td>
</tr>
<tr>
<td>Heterogeneous clusters</td>
<td>PG &gt;=7.4 Unix only</td>
<td>PG &gt;=7.4 Unix only</td>
<td>PG</td>
<td>PG</td>
<td>PG&gt;=7.3.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Geo-cluster support</td>
<td>No</td>
<td>No</td>
<td>Possible but don’t use</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Performance vs Scalability

/ **Performance**
  - latency different from throughput

/ **Most solutions don’t provide parallel query execution**
  - No parallelization of query execution plan
  - Query do not go faster when database is not loaded

/ **What a perfect load distribution buys you**
  - Constant response time when load increases
  - Better throughput *when* load surpasses capacity of a single database
Understanding scalability (1/2)

Performance vs. Time

- 1 Database - Load in users
- 1 Database - Response time
- Sequoia 2 DBs - Load in users
- Sequoia 2 DBs - Response time

Single DB

Sequoia

20 users
Understanding scalability (2/2)

Performance vs. Time

1 DB - Load in users
1 DB - Response time
Sequoia 2DB - Load in users
Sequoia 2 DB - Response time

Single DB
Sequoia
90 users
**RAIDb Concept:** Redundant Array of Inexpensive Databases

- RAIDb controller – creates single virtual db, balances load
- RAIDb 0,1,2: various performance/fault tolerance tradeoffs
- New combinations easy to implement

### RAIDb-0
- partitioning (whole tables)
- no duplication
- no fault tolerance
- at least 2 nodes

### RAIDb-1
- mirroring
- performance bounded by write broadcast
- at least 2 nodes
- uni/cluster certifies only RAIDb-1

### RAIDb-2
- partial replication
- at least 2 copies of each table for fault tolerance
- at least 3 nodes
Sequoia architectural overview

/ **Middleware implementing RAIDb**
  - 100% Java implementation
  - open source (Apache v2 License)

/ **Two components**
  - Sequoia driver (JDBC, ODBC, native lib)
  - Sequoia Controller

/ **Database neutral**
Sequoia read request

Client application
(Servlet, EJB, ...)

Sequoia driver

get connection from pool

update cache
(if available)

Client application
(Servlet, EJB, ...)

Sequoia driver

exec
RR, WRR, LPRF, ...

ordering

execute

SELECT * FROM t

login, password

connect
myDB
Sequoia write request

jdbc:sequoia://node1,node2/myDB

Total order reliable multicast
Alternative replication algorithms

/ **GORDA API**
  - European consortium defining API for pluggable replication algorithms

/ **Sequoia 3.0 GORDA compliant prototype for PostgreSQL**
  - Uses triggers to compute write-sets
  - Certifies transaction at commit time
  - Propagate write-sets to other nodes

/ **Tashkent/Tashkent+**
  - Research prototype developed at EPFL
  - Uses workload information for improved load balancing

/ **More information**
  - http://sequoia.continuent.org
  - http://gorda.di.uminho.pt/
PostgreSQL specific issues

/ Indeterminist queries
  • Macros in queries (now(), current_timestamp, rand(), …)
  • Stored procedures, triggers, …
  • SELECT … LIMIT can create non-deterministic results in UPDATE statements if the SELECT does not have an ORDER BY with a unique index:
    UPDATE FOO SET KEYVALUE='x' WHERE ID IN (SELECT ID FROM FOO WHERE KEYVALUE IS NULL LIMIT 10)

/ Sequences
  • setval() and nextval() are not rollback
  • nextval() can also be called within SELECT

/ Serial type

/ Large objects and OIDs

/ Schema changes

/ User access control
  • not stored in database (pg_hba.conf)
  • host-based control might be fooled by proxy
  • backup/restore with respect to user rights

/ VACUUM
Outline

/ Database replication strategies
/ PostgreSQL replication solutions
/ Building HA solutions
/ Management issues in production
Simple hot-standby solution (1/3)

- Virtual IP address + Heartbeat for failover
- Slony-I for replication
Simple hot-standby solution (2/3)

- Virtual IP address + Heartbeat for failover
- Linux DRDB for replication
- Only 1 node serving requests
Simple hot-standby solution (3/3)

/ pgpool for failover
/ proxy might become bottleneck
  • requires 3 sockets per client connection
  • increased latency
/ Only 1 node serving requests

Client Applications

pgpool

Postgres 1

Postgres 2
Highly available web site

- **Apache clustering**
  - L4 switch, RR-DNS, One-IP techniques, LVS, Linux-HA, …

- **Web tier clustering**
  - mod_jk (T4), mod_proxy/mod_rewrite (T5), session replication

- **PostgreSQL multi-master clustering solution**
Highly available web applications

- Consider MTBF (Mean time between failure) of every hardware and software component
- Take MTTR (Mean Time To Repair) into account to prevent long outages
- Tune accordingly to prevent trashing
Building Geo-Clusters

America master
Europe slave
Asia slave

America slave
Europe master
Asia slave

asynchronous
WAN replication

America slave
Europe slave
Asia master

Database

Database
This is what you should NOT do:
- At least 2 network adapters in controller
- Use a dedicated network for controller communication
Split brain problem (2/2)

- When controllers lose connectivity clients may update inconsistently each half of the cluster.
- No way to detect this scenario (each half thinks that the other half has simply failed).
Avoiding network failure and split-brain

- Collocate all network traffic using Linux Bonding
- Replicate all network components (mirror the network configuration)
- Various configuration options available for bonding (active-backup or trunking)
Synchronous GeoClusters

- Multi-master replication requires group communication optimized for WAN environments
- Split-brain issues will happen unless expensive reliable dedicated links are used
- Reconciliation procedures are application dependent
Outline

/ Database replication strategies
/ PostgreSQL replication solutions
/ Building HA solutions
/ Management issues in production
Managing a cluster in production

/ Diagnosing reliably cluster status

/ Getting proper notifications/alarms when something goes wrong
  • Standard email or SNMP traps
  • Logging is key for diagnostic

/ Minimizing downtime
  • Migrating from single database to cluster
  • Expanding cluster
  • Staging environment is key to test

/ Planned maintenance operations
  • Vacuum
  • Backup
  • Software maintenance (DB, replication software, …)
  • Node maintenance (reboot, power cycle, …)
  • Site maintenance (in GeoCluster case)
Dealing with failures

/ Software vs Hardware failures
  • client application, database, replication software, OS, VM, …
  • power outage, node, disk, network, Byzantine failure, …
  • Admission control to prevent trashing

/ Detecting failures require proper timeout settings

/ Automated failover procedures
  • client and cluster reconfiguration
  • dealing with multiple simultaneous failures
  • coordination required between different tiers or admin scripts

/ Automatic database resynchronization / node repair

/ Operator errors
  • automation to prevent manual intervention
  • always keep backups and try procedures on staging environment first

/ Disaster recovery
  • minimize data loss but preserve consistency
  • provisioning and planning are key

/ Split brain or GeoCluster failover
  • requires organization wide coordination
  • manual diagnostic/reconfiguration often required
Summary

/ Different replication strategies for different needs
/ Performance ≠ Scalability
/ Manageability becomes THE major issue in production
Links

/ **pgpool**: http://pgpool.projects.postgresql.org/
/ **PGcluster**: http://pgcluster.projects.postgresql.org/
/ **Slony**: http://slony.info/
/ **Sequoia**: http://sequoia.continuent.org
/ **GORDA**: http://gorda.di.uminho.pt/
/ **Slides**: http://sequoia.continuent.org/Resources

http://www.continuent.org
RAIDb-2 for scalability

- limit replication of heavily written tables to subset of nodes
- dynamic replication of temp tables
- reduces disk space requirements
RAIDb-2 for heterogeneous clustering

/ Migrating from MySQL to Oracle
/ Migrating from Oracle x to Oracle x+1

Client program

Sequoia driver

Sequoia controller

MySQL driver

Oracle driver

MySQL Old tables

Oracle migrated tables

MySQL driver

Oracle driver

MySQL Old tables

Oracle migrated + new apps

Oracle 11h driver

Oracle new apps

Oracle new apps
Server farms with master/slave db replication

- No need for group communication between controller
- Admin. operations broadcast to all controllers
Composing Sequoia controllers

- Sequoia controller viewed as single database by client (app. or other Sequoia controller)
- No technical limit on composition deepness
- Backends/controller cannot be shared by multiple controllers
- Can be expanded dynamically