PostgreSQL HA Database Clusters through Containment

Le Quan Ha
IPG Database Team,
BlackBerry RIM,
176 Columbia St. W., Waterloo, ON, Canada N2L 3L3
403-828-1846
quan-ha.le@tenzing.com
OVERVIEW OF THE REPORT

- PART 1: THEORETICAL MODELS
- PART 2: IMPLEMENTATION
- PART 3: PERFORMANCE ANALYSIS
- CONCLUSIONS
PART 1: THEORETICAL MODELS

• What is keepalived-repmgr cluster?
• What is HAProxy-PgBouncer cluster?
• Same containment
• Keepalived: VRRP is a fundamental brick for failover
• Repmgr: an open-source tool suite to manage replication in a cluster of PostgreSQL servers
• Failover:
  - Master fails, keepalived will switch the virtual IP to the hot standby
  - Hot standby's VRRP instance of keepalived changes to MASTER state
  - notify_master script is automatically called to promote the hot standby to be a new master
HAPROXY-PGBOUNCER

• HAPerxy (High Availability Proxy): an open source software TCP/HTTP Load Balancer and proxying solution

• PgBouncer: a lightweight connection pooler for PostgreSQL
  ➢ Three modes of pooling: session pooling, transaction pooling and statement pooling.
  ➢ Low memory requirements

• The Frontend servers are inside the same subnet
• Cross-containment: The Backend servers can be in different subnets
• Load balancing: distributing the workload across multiple computing resources

• Shared witness server in clusters: to avoid a "split-brain" situation and control / decide to failover to a privilege standby
HAPROXY-PGBOUNCER

- Cross-containment
- Load Balancing
- Connection Pooler
- Replication
- Healthcheck
- Failover / WITNESS

FRONTEND CLUSTER

Application

Keepalived virtual IP

HAPROXY 1

Load Balancer 1
PgBouncer

PostgreSQL Master

replication

Load Balancer 2
PgBouncer

PostgreSQL Hot Standby

Load Balancer 3
PgBouncer

PostgreSQL Async Standby

PostgreSQL Shared WITNESS

Failover Decision

BACKEND CLUSTER
PART 2: IMPLEMENTATION

• Development of keepalived-repmgr clusters

• Research and development of HAProxy-PgBouncer cluster
  ➢ Flow of Read Requests
  ➢ Flow of Write Requests
  ➢ Statistics Report
  ➢ Farm Failover
  ➢ Auto Failover
  ➢ Frontend cluster: keepalived
  ➢ Backend cluster: PgBouncer
  ➢ Shared WITNESS
  ➢ Switchback

• Development of the Shared WITNESS between 2 different clusters
KEEPALIVED-REPMGR

• Altus cloud: 20 network zones (16 productions zones, 2 laboratory zones and 2 restricted pre-production zones)
• SATURN RING software: on 15 production zones
  ➢ 10/02 GE North container 2
  ➢ 10/03 GE North container 3
  ➢ 11/02 GE South container 2
  ➢ 11/03 GE South container 3
  ➢ 5/02 Spirit East container 2
  ➢ 5/03 Spirit East container 3
  ➢ 6/02 Spirit West container 2
  ➢ 6/03 Spirit West container 3
  ➢ 7/01 Viking Container 1
  ➢ ONELAB - Orion
  ➢ ONELAB - Thor
  ➢ 21/01 Stardust
  ➢ Casino
  ➢ HongKong
  ➢ Mercury
Our executable actual implementation from the theoretical model
FLOW OF READ REQUESTS

FRONTEND CLUSTER

- Empty PostgreSQL
- HAPerxy
- Keepalived

BACKEND CLUSTER

- PostgreSQL
- repmgr
- repmgrd
- PgBouncer

- Master VM: PGHADB1 10.236.49.18
- Hot Standby VM: PGHADB2 10.236.122.183
- Async Standby VM: PGHADB3 10.236.49.19
- Shared Witness VM: PGWITNESS 10.236.134.191
FLOW OF WRITE REQUESTS

FRONTEND CLUSTER
- Empty PostgreSQL
- HAProxy
- Keepalived

BACKEND CLUSTER
- PostgreSQL
- repmgr
- repmgrd
- PgBouncer

PGHAPROXY VM1
- Port 6433
- 10.236.134.192

PGHABACKUP VM2
- Port 6432
- 10.236.134.193

Master VM3
- PGHADB1
- 10.236.49.18

Hot Standby VM4
- PGHADB2
- 10.236.122.183

Async Standby VM5
- PGHADB3
- 10.236.49.19

Shared Witness VM6
- PGWITNESS
- 10.236.134.191

Statistics can be defined as

```
listen stats 0.0.0.0:8080
mode http
stats enable
stats uri /HAproxy?stats
stats realm Strictly\ Private
stats auth admin:admin
```
FARM Failover of HAProxy

- When the master database fails
- The Hot Standby will be promoted to be the new master

BACKEND CLUSTER

Master fails
FARM Failover – Read Requests

FRONTEND

Virtual IP
10.236.134.194

PORT 6432

PGHAPROXY VM
10.236.134.192

PGHABACKUP VM
10.236.134.193

BACKEND

• At first Master fails
• Then a Read will performs from the new promoted Hot Standby (new master) and Async Standby

PORT 5432

Hot Standby VM
PGHADB2
10.236.122.183

Async Standby VM
PGHADB3
10.236.49.19

Shared Witness VM
PGWITNESS
10.236.134.191

failover messages
- When the Master fails, Writes also performs to the new promoted master (Hot Standby) only
- The Async Standby receives log files to sync its Database
FRONTEND CLUSTER

Keepalived

Virtual IP 10.236.134.194

PGHAPROXY VM$_1$
10.236.134.192

PGHABACKUP VM$_2$
10.236.134.193

- Keepalived understands that the 2 VMs are sharing one instance
- There should be one Master VM and one Backup VM
- A virtual IP is defined
- The Master will keep the virtual IP by default
- If the Master fails, timing control to switch the virtual IP for the Backup VM
BACKEND CLUSTER: PgBouncer

- PgBouncer are installed on Master, Hot Standby and Async Standby of the Backend
- Connection Pooler by PgBouncer
- Receiving requests from PgBouncer port 6432
- Forward to port 5432
Shared WITNESS

- The WITNESS is shared between the current HAProxy-PgBouncer cross-containment cluster and a second Keepalived-repmgr cluster that is not cross-containment.

**HAProxy-PgBouncer cluster**

- **Virtual IP**: 10.236.134.190
- **Shared WITNESS**: 10.236.134.190

**Keepalived-repmgr cluster**

- **Virtual IP**: 10.236.134.190
- **Async Standby**: PGASYNCSTANDBY 10.236.134.189
- **Hot Standby**: PGHOTSTANDBY 10.236.134.188
- **Shared Witness**: PGWITNESS 10.236.134.191
Shared WITNESS

- In Witness, there are 2 different repmgr.conf config files for 2 clusters
  priority=-1
- HAProxy-PgBouncer
  /db/postgres_config/main/repmgr.conf
- Keepalived-repmgr
  /db/postgres_config/main/repmgr/witness/repmgr.conf

- How to create

  postgres@PGWITNESS:~$ repmgr -d postgres -U repmgr -h 10.236.49.18
  -D /var/lib/postgresql/9.4/main -f /db/postgres_config/main/repmgr.conf
  witness create --force --verbose

  postgres@PGWITNESS:~$ repmgr -d postgres -U repmgr -h 10.236.134.187
  -D /var/lib/postgresql/9.4/main -f /db/postgres_config/main/repmgr/witness/repmgr.conf
  witness create --force --verbose
How to show 2 different clusters on the same WITNESS

```
root@PGWITNESS:~# repmgr -f /db/postgres_config/config/repmgr.conf cluster show
Role       | Connection String
* master   | host=10.236.134.187 user=repmgr password=passw0rd dbname=postgres
witness    | host=10.236.134.191 user=repmgr password=passw0rd dbname=postgres
standby    | host=10.236.122.183 user=repmgr password=passw0rd dbname=postgres
standby    | host=10.236.49.19 user=repmgr password=passw0rd dbname=postgres
```

```
root@PGWITNESS:~# repmgr -f /db/postgres_config/config/repmgr/witness/repmgr.conf cluster show
Role       | Connection String
* master   | host=10.236.134.187 user=repmgr password=passw0rd dbname=postgres
witness    | host=10.236.134.191 user=repmgr password=passw0rd dbname=postgres
standby    | host=10.236.122.183 user=repmgr password=passw0rd dbname=postgres
standby    | host=10.236.49.19 user=repmgr password=passw0rd dbname=postgres
```
Switchback for the HAProxy-PgBouncer cluster

BACKEND CLUSTER:
• repmgr clone the failed Master to the new promoted master (Hot Standby)
• Start service on the failed Master
• Follow the new promoted master on the failed Master
  ➢ THE FAILED MASTER IS NOW RESTARTED AS A NEW STANDBY
• Stop service on the new promoted master (Hot Standby)
• Promote the failed Master to be back to Master node
  ➢ THE FAILED MASTER IS NOW BACK AS MASTER AGAIN
• repmgr clone the Hot Standby to the Master
• Start service on the Hot Standby
• Follow the Master on the Hot Standby
THE NEW PROMOTED MASTER (HOT STANDBY) IS NOW BACK AS HOT STANDBY AGAIN
• Follow the Master on the Async Standby (and any other Standbys)

FRONTEND CLUSTER
• Restart haproxy service on HAProxy-1 to return haproxy’s load balancing setting
PART 3: PERFORMANCE ANALYSIS

• Methodology

• Keepalived-repmgr Throughputs
• Keepalived-repmgr I/O and CPU Graphs
• Keepalived Failover CPU Graph

• HAPerxy-PgBouncer Throughputs
• HAPerxy-PgBouncer Load Balancing
• HAPerxy-PgBouncer I/O and CPU Graphs
• HAPerxy-PgBouncer Failover CPU Graphs

• Performance Comparison: keepalived-repmgr vs. HAPerxy-PgBouncer
METHODOLOGY

• Apache JMeter v2.13 to create test plans of 1 million samples/each

\[
Throughput = \frac{\text{Number of Transactions}}{\text{Real Execution in seconds}} \tag{1}
\]

\[
KB/sec = \frac{\text{Throughput} \times \text{Avg. Bytes}}{1024} \tag{2}
\]

• 6 performance tests by HTTP Requests:
  - Read Only without data execution
  - Read Only with data execution
  - Simple Write with Inserts and Updates
  - Simple Write with Deletes
  - Read Write with Selects, Inserts and Updates
  - Read Write with Selects and Deletes.

• Each performance test, we report 8 graphs for
  - Transactions per Second
  - CPU Usages
  - Active Threads
  - Response Time
  - Bytes Throughput over Time
  - Response Times Percentiles
  - Response Times vs Threads
  - Transaction Throughput vs Threads
## KEEPALIVED-REPMGR THROUGHPUT

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Test duration</th>
<th>Avg. Response Time /sample</th>
<th>Throughput</th>
<th>KB/sec</th>
<th>Avg. Bytes /transaction</th>
<th>Avg. Latency /transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only without data execution</td>
<td>463.976s</td>
<td>27.494s</td>
<td>2,155.284</td>
<td>609.467</td>
<td>289.565</td>
<td>27.443</td>
</tr>
<tr>
<td>Read Only with data execution</td>
<td>478.775s</td>
<td>29.951s</td>
<td>2,088.664</td>
<td>8,063.858</td>
<td>3,953.432</td>
<td>29.909</td>
</tr>
<tr>
<td>Simple Write with Inserts and Updates</td>
<td>753.529s</td>
<td>58.480s</td>
<td>1,327.089</td>
<td>376.119</td>
<td>290.219</td>
<td>58.418</td>
</tr>
<tr>
<td>Simple Write with Deletes</td>
<td>533.122s</td>
<td>31.481s</td>
<td>1,875.743</td>
<td>530.419</td>
<td>289.565</td>
<td>31.421</td>
</tr>
<tr>
<td>Read Write with Selects, Inserts and Updates</td>
<td>981.059s</td>
<td>80.431s</td>
<td>1,019.307</td>
<td>288.666</td>
<td>289.995</td>
<td>80.372</td>
</tr>
<tr>
<td>Read Write with Selects and Deletes</td>
<td>570.773s</td>
<td>37.486s</td>
<td>1,752.010</td>
<td>493.719</td>
<td>288.565</td>
<td>37.426</td>
</tr>
</tbody>
</table>
Keepalived-repmgr: Read Write with Selects, Inserts and Updates - Transactions per Second
Keepalived-repmgr: Read Write with Selects and Deletes - CPU Usages
Keepalived-repmgr: Simple Write with Deletes
- Active Threads
Keepalived-repmgr: Read Only with Data Execution
- Response Time
Keepalived-repmgr: Simple Write with Inserts and Updates - Bytes Throughput over Time
Keepalived-repmgr: Read Only without Data Execution
- Response Times Percentiles
Keepalived-repmgr: Read Write with Selects and Deletes
- Response Times vs Threads
Keepalived-repmgr: Simple Write with Inserts and Updates - Transaction Throughput vs Threads

![Graph showing transaction throughput vs number of active threads](image-url)
Keepalived-repmgr: Failover CPU usages

![Graph showing CPU usage over time for different roles: Master, Hot Standby, Async Standby, and Witness. The graph displays peaks and troughs indicating CPU usage fluctuations.](image-url)
<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Test duration</th>
<th>Avg. Response Time /sample</th>
<th>Throughput</th>
<th>KB/sec</th>
<th>Avg. Bytes /transaction</th>
<th>Avg. Latency /transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only without data execution</td>
<td>423.901s</td>
<td>28.234s</td>
<td>2,359.041</td>
<td>662.477</td>
<td>287.565</td>
<td>28.228</td>
</tr>
<tr>
<td>Read Only with data execution</td>
<td>471.192s</td>
<td>28.215s</td>
<td>2,122.277</td>
<td>1,354.034</td>
<td>653.322</td>
<td>28.209</td>
</tr>
<tr>
<td>Simple Write with Inserts and Updates</td>
<td>702.484s</td>
<td>55.893s</td>
<td>1,423.520</td>
<td>1,895.078</td>
<td>1,363.212</td>
<td>55.886</td>
</tr>
<tr>
<td>Simple Write with Deletes</td>
<td>521.546s</td>
<td>36.755s</td>
<td>1,917.376</td>
<td>540.319</td>
<td>288.565</td>
<td>36.749</td>
</tr>
<tr>
<td>Read Write with Selects, Inserts and Updates</td>
<td>970.949s</td>
<td>77.564s</td>
<td>1,029.920</td>
<td>679.687</td>
<td>675.780</td>
<td>77.557</td>
</tr>
<tr>
<td>Read Write with Selects and Deletes</td>
<td>568.803s</td>
<td>42.116s</td>
<td>1,758.078</td>
<td>626.928</td>
<td>365.157</td>
<td>42.110</td>
</tr>
</tbody>
</table>
SYSSTAT is showing that the Reads are shared between Backend Master, Hot Standby and Async Standby
HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Transactions per Second
HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Frontend CPU usages
HAProxy - PgBouncer: Read Only with Data Execution
- Backend database server CPU usages on Load Balancing
HAProxy-PgBouncer: Read Only without Data Execution
- Active Threads

![Graph showing Active Threads over time]
HAProxy-PgBouncer: Read Write with Selects and Deletes - Response Time
HAProxy-PgBouncer: Simple Write with Deletes
- Bytes Throughput over Time
HAProxy-PgBouncer: Simple Write with Deletes
- Response Times Percentiles
HAProxy-PgBouncer: Simple Write with Inserts and Updates
- Response Times vs Threads
HAProxy-PgBouncer: Simple Write with Inserts and Updates
- Transaction Throughput vs Threads

![Graph showing transaction throughput vs number of active threads. The graph peaks at approximately 1,819.392 transactions per second with 55 active threads.]
HAProxy-PgBouncer: Failover Front-end CPU usages
HAProxy-PgBouncer: Failover Back-end CPU usages
## PERFORMANCE COMPARISON: 
**KEEPALIVED-REPMGR vs. HAPROXY-PGBOUNCER**

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Throughput improvement of HAProxy-PgBouncer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Only without data execution</td>
<td>9.454%</td>
</tr>
<tr>
<td>Read Only with data execution</td>
<td>1.609%</td>
</tr>
<tr>
<td>Simple Write with Inserts and Updates</td>
<td>7.266%</td>
</tr>
<tr>
<td>Simple Write with Deletes</td>
<td>2.220%</td>
</tr>
<tr>
<td>Read Write with Selects, Inserts and Updates</td>
<td>1.041%</td>
</tr>
<tr>
<td>Read Write with Selects and Deletes</td>
<td>0.346%</td>
</tr>
</tbody>
</table>
CONCLUSIONS

- HAProxy-PgBouncer cluster supplies good cross-containment approach
- The IPG, Database Group have achieved good purposes
  - Load Balancing
  - Farm Failover
  - Healthcheck
  - Auto-Failover
- Performance Analysis have been done by JMeter HTTP Requests combined with PhP using Fast CGI on Apache2
  - Read Only
  - Simple Write
  - Read Write
- HAProxy-PgBouncer improves the throughputs from 0.346% to 9.454% performance than keepalived-repmgr
- Keepalived-repmgr also does not offer cross-containment and load balancing abilities
- HAProxy-PgBouncer also provides two different methods to implement failovers: auto-failover and farm-failover