PostgreSQL HA Database Clusters through Containment

Le Quan Ha
Infrastructure Platforms, Database Group, BlackBerry, Waterloo, Ontario N2L 3L3, Canada
Telephone: +1-403-828-1846, Email: NLp.Sr@Shaw.ca

Abstract: The enormous amount of data flow has made Relation Database Management System the most important and popular tools for persistence of data. While open-source RDBMS systems are not as widely used as proprietary systems like Oracle databases, but over the years, systems like PostgreSQL have gained massive popularity. High-availability database clusters (also known as HA clusters or failover clusters) are groups of computer systems that support server applications that can be reliably utilized with a minimum of down-time. This article is an attempt to set a benchmark of PostgreSQL high-availability databases in comparing the performance of same-containment keepalived-repmgr clusters against cross-containment HAProxy-PgBouncer clusters. The result shows that our cross-containment HAProxy-PgBouncer is still a significantly better performer with load balancing, healthcheck and its throughput are improved from 0.346% to 9.454% in compared to keepalived-repmgr. Also we would like to present I/O activities and CPU usage percentages between the two kinds of PostgreSQL HA clusters.

Keywords: Altus cloud, cross containment, failover, HAProxy, healthcheck, high availability, keepalived, load balancing, PgBouncer, PostgreSQL, repmgr, repmgred.

I. INTRODUCTION

We are developing the Altus cloud that includes of around 20 network zones (16 productions zones, 2 laboratory zones and 2 restricted pre-production zones.) The Altus cloud stores information of various styles of databases such as Oracle, PostgreSQL, Apache Cassandra, MariaDB/MySQL, Elasticsearch and MS SQL*Server. The PostgreSQL databases are currently ranked the 4th most popular according to DB-Engines in 2015 [14]; it was earlier ranked the 6th position by Emison, 2014 [16]. The PostgreSQL databases are developing into high-availability clusters on around 15 zones of Saturn Ring software, a storage system designed in such a way that the Saturn staffs as well as the organization’s users feel comfortable in using the software.

We saw the development of this project as an opportunity for analysing the comparative performance of PostgreSQL databases that are developing by keepalived-repmgr clusters on the Saturn Ring software and the other research-and-development PostgreSQL databases that are built by HAProxy-PgBouncer cross-containment cluster. The main focus of this paper is to analyse the performance of the two kinds of systems namely Saturn Ring keepalived-repmgr and R&D HAProxy-PgBouncer.

II. WHY KEEPALIVED-REPMGR AND HAPROXY-PGBOUNCER

Keepalived-repmgr and HAProxy-PgBouncer are two of the most popular high availability database clusters. Keepalived-repmgr is the configuration for database systems on the same containment (using one same subnet for all of the database servers, thus on one same network zone and same gateway,) while HAProxy-PgBouncer is the cross-containment cluster in which back-end database servers can be arranged on different containments by different network zones thus each database server can be assigned a different subnet and communicates to each other through different gateways.

Keepalived-repmgr and HAProxy-PgBouncer are the configurations that were selected based on the convenience of the developers, available resources and the fact that the expected project happens to use one same containment or cross-containments in the database.

While comparing between Keepalived-repmgr and HAProxy-PgBouncer architectures, we would like to present our research of performance analysis in cloud computing with the open-source PostgreSQL RDBMS.

A. Keepalived-repmgr

Keepalived is a routing software written in C. The main goal of keepalived is to provide simple and robust facilities for high-availability to our system and infrastructures. High-availability is achieved employing VRRP protocol. VRRP is a fundamental brick for failover. In addition, keepalived implements a set of hooks to the VRRP finite state machine providing low-level with high-speed protocol interactions.

Repmgr is an open-source tool suite to manage replication in a cluster of PostgreSQL servers. It provides tools to set up standby servers, monitor replication, and perform administrative tasks such as failover or manual switchover operations. The repmgr tool has provided advanced support for replication mechanisms.

B. HAProxy-PgBouncer

HAProxy (High Availability Proxy), is a popular open source software TCP/HTTP Load Balancer and proxying solution which can be run on Linux, Solaris, and FreeBSD. Its most common use is to improve the performance and
reliability of a database cluster environment by distributing the workload across multiple computing resources. It is used in many high-profile environments GitHub, Imgur, Instagram, and Twitter.

PgBouncer is a lightweight connection pooler for PostgreSQL; there are three modes of pooling: session pooling, transaction pooling and statement pooling. PgBouncer has low memory requirements (by default 2K per connection). This is due to the fact that PgBouncer does not need to see full packet at once. It is not tied to one backend server, the destination databases can reside on different hosts.

III. RELATED WORK

Database systems have strongly developed nonstop since 1980s through various authors Khoshafian, Copeland, Jagodis, Boral and Valduriez, 1987 [29]; Abiteboul, Hull and Vianu, 1995 [27]; Boncz and Kersten, 1999 [22]; Abadi, Madden and Ferreira, 2006 [11]; Abadi, Marcus, Madden and Hollenbach, 2007 [8, 9]; Abadi, Myers, DeWitt and Madden, 2007 [10].

Many performance analysis researches between different kinds of database systems have been developed for database theory: Ailamaki, DeWitt, Hill and Skoumakis, 2001 [1]; Dwivedi, Lamba and Shukla (2012) [2].

Performance of the most popular databases are compared in timing durations of queries, the CPU usages and memory costs. In 2011, Bassil measured through MS SQL Server 2008, Oracle 11g, IBM DB2, MySQL 5.5, and MS Access 2010 [36]. Lee (2013) compared performances of Oracle, MySQL and SQL Server [17].

Semantic web databases that have been characterized by read/write performances of RDF [18, 24, 25, 26] through the Welcome project’s data [33]; of the Virtuoso Universal Server 6 Open Source Edition of Erling and Mikhailov, 2009 [21]; of JENA [4, 15, 31]; of SPARQL by the BSBM [5] and Bizer and Schultz, 2009 [6]... are also analyzed for performance by Guo, Pan and Hefflin, 2005 [37].

A special group of databases that provide a different model for storage and retrieval of data from the tabular relations, are NoSQL databases such as Apache Cassandra, HBase and MongoDB. Their performances are compared to each other by Datastax Corporation, 2013 and 2014 [12, 13]; Gansen, Huang, Liang and Tang, 2013 [38]... and they are also compared to open-source MySQL database by Gupta and Narasima, 2015 [28].

Recently, cloud computing with databases has evolved as a new computing paradigm, allowing end users to utilise the resources on a demand-driven basis, unlike grid and cluster computing which are the traditional approaches to access resources. It is characterised by the 4V’s, such as Volume, Velocity, Veracity and Variety by Gudivada, Rao and Raghavan, 2014 [35]; Sandholm and Lee, 2014 [32]; Agrawal, Das and Abbadi, 2011 [7]; Naim, Yassin, Zamri and Sarmin, 2011 [20]; and Vora, 2011 [19]...

Open-source RDBMS has been researched using MySQL by Saikia, Joy, Dolma and Mary. R (2015) [3] and Kilintzis, Beredimas and Chouvarda (2014) [34]; using Db4o by Kulshrestha and Sachdeva (2014) [30]... and more performance analysis was done for less popular open-source databases such as MonetDB by Boncz, Zukowski and Nes (2005) [23].

IV. PROPOSED METHODOLOGY

A. Keepalived-repmgr theoretical model

Fig 1 shows our theoretical model of keepalived-repmgr; when the master fails, keepalived will switch the virtual IP to the hot standby. At this time, the hot standby’s VRRP instance of keepalived changes to MASTER state and a notify_master script is automatically called to promote the hot standby to be a new master.

There is a shared witness server in the cluster, it is important to avoid a “split-brain” situation and control / decide to failover to a privilege standby. The witness server is essential to ensure that one network segment has a “voting majority”, so other segments will know they are in the minority and not attempt to promote a new master.

A witness server can be set up using repmgr witness create and can run on a dedicated server or an existing node.

B. HAProxy-PgBouncer theoretical model

Fig 2 shows our theoretical model of the HAProxy-PgBouncer cluster with load balancing.

When HAProxy-PgBouncer receives read requests from the application, it has load balancing capability that distributes these read-loads across multiple back-end database servers. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload errors; it may increase reliability and availability through redundancy.
On the front-end, when HAProxy-1 server fails, keepalived will switch the virtual IP to HAProxy-2. On the back-end, when the master database fails, repmgrd (replication manager watchdog) will promote the hot standby to be new master and a failover happens.

In HAProxy-PgBouncer, it makes sense to create a witness server in conjunction with running repmgrd.

V. RESULTS

We use Apache JMeter v2.13 to create test plans of 1 million samples/each with the main formulae given below:

\[ \text{Throughput} = \frac{\text{Number of Transactions}}{\text{Real Execution in seconds}} \]  \hspace{1cm} (1)

\[ KB/\text{sec} = \frac{\text{Throughput} \times \text{Avg. Bytes}}{1024} \]  \hspace{1cm} (2)

There are 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; and Read Write with Selects and Deletes.

For 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

Altogether we obtained 48 performance charts for keepalived-repmgr and 54 for HAProxy-PgBouncer, we would like to report typical charts only. Then for each cluster, we also observe the CPU usages of failovers.

A. Throughput performance of keepalived-repmgr

With further developments in replication functionality such as cascading replication, timeline switching and base backups via the replication protocol, the team has decided to use PostgreSQL 9.4.1 and repmgr 3.0.1; the version of keepalived is 1.2.16. All of the servers are on Ubuntu 14.04.1 LTS.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>KEEPALIVED-REPMGR / JMETEER</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP Request</strong></td>
<td><strong>Test duration</strong></td>
</tr>
<tr>
<td>Read Only without data execution</td>
<td>463.976s</td>
</tr>
<tr>
<td>Read Only with data execution</td>
<td>478.775s</td>
</tr>
<tr>
<td>Simple Write with Inserts and Updates</td>
<td>753.529s</td>
</tr>
<tr>
<td>Simple Write with Deletes</td>
<td>533.122s</td>
</tr>
<tr>
<td>Read Write with Selects, Inserts and Updates</td>
<td>981.059s</td>
</tr>
<tr>
<td>Read Write with Selects and Deletes</td>
<td>570.773s</td>
</tr>
</tbody>
</table>
We applied the keepalived-repmgr model for the Saturn Ring software (written by python Django) for all over 15 production network zones on Altus cloud. Our keepalived-repmgr throughputs are reported by Table I – 0% error rates for all 6 million requests (no errors) and performance charts are shown from Fig 3 to Fig 10.

Fig. 3. Keepalived-repmgr: Read Write with Selects, Inserts and Updates - Transactions per Second

Fig. 4. Keepalived-repmgr: Read Write with Selects and Deletes - CPU Usages

Fig. 5. Keepalived-repmgr: Simple Write with Deletes - Active Threads

Fig. 6. Keepalived-repmgr: Read Only with Data Execution - Response Time

Fig. 7. Keepalived-repmgr: Simple Write with Inserts and Updates - Bytes Throughput over Time

Fig. 8. Keepalived-repmgr: Read Only without Data Execution - Response Times Percentiles

Fig. 9. Keepalived-repmgr: Read Write with Selects and Deletes - Response Times vs Threads

Fig. 10. Keepalived-repmgr: Simple Write with Inserts and Updates - Transaction Throughput vs Threads

A CPU usage chart for Failover is shown in Fig 11 as below, in which the hot standby is promoted to be the new master.
B. Throughput performance of HAProxy-PgBouncer clusters

We developed a cross-containment HAProxy-PgBouncer cluster through the two separate network zones of the Altus cloud. The HAProxy-1, HAProxy-2, hot standby and witness servers are on the first laboratory zone while the master and the async standby are on the second zone. The tool versions are PostgreSQL 9.4.1, HAProxy 1.5.11, keepalived 1.2.16, PgBouncer 1.5.5 and repmgr 3.0.1. All of the servers are on Ubuntu 14.04.1 LTS.

Fig 12 shows our real implementation of the load balancing for read requests with real observation shown in Fig 13 by sysstat to verify that the read-loads are balanced through back-end master, hot standby and async standby. Performance are shown from Fig 14 to Fig 22 and the throughput summary in Table II. For altogether 6 million requests, error rates are 0%.

![Fig. 11. Keepalived-repmgr: Failover CPU usages](image)

![Fig. 12. Implementation of HAProxy-PgBouncer cluster](image)

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Test duration</th>
<th>Avg. Response Time/s</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>
Fig. 13. Load balancing observed by sysstat on the master PGHADB1, the hot standby PGHADB2 and the async standby PGHADB3.

Fig. 14. HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Transactions per Second.

Fig. 15. HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Frontend CPU usages.

Fig. 16. HAProxy-PgBouncer: Read Only with Data Execution - Backend database server CPU usages on Load Balancing.

Fig. 17. HAProxy-PgBouncer: Read Only without Data Execution - Active Threads.

Fig. 18. HAProxy-PgBouncer: Read Write with Selects and Deletes - Response Time.

Fig. 19. HAProxy-PgBouncer: Simple Write with Deletes - Bytes Throughput over Time.
In order to process failovers when the master database fails, both of the HAProxy farm failover mechanism and the HAProxy auto-failover mechanism are set up. The farm failover mechanism for HAProxy is applied to make the read requests transparently continuous during master failures. In order to continue write requests transparently through master failures, HAProxy auto-failover mechanism by healthcheck services using xinetd (extended Internet daemon) are installed on port 5678 of all back-end master, hot standby and async standby.

HAProxy-PgBouncer also provides two different methods to implement failovers: auto-failover and farm-failover.

### VI. COMPARISON AND DISCUSSION

HAProxy-PgBouncer does not only include better cross-containment and load balancing features than the keepalived-repmgr cluster, but also the throughputs of HAProxy-PgBouncer cluster are improved from 0.346% to 9.454% in compared to the keepalived-repmgr cluster as in Table III.

#### TABLE III

<table>
<thead>
<tr>
<th>HTTP Request</th>
<th>Throughput improvement</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>

### VII. CONCLUSIONS

The purpose of this paper is to analyse the performance of two popular high availability PostgreSQL clusters, keepalived-repmgr and HAProxy-PgBouncer, in terms of transaction throughputs. The results show that HAProxy-PgBouncer improves the throughputs from 0.346% to 9.454% performance than keepalived-repmgr when executing one million HTTP requests from JMeter. In all the test cases, the numbers of transactions per second of HAProxy-PgBouncer are higher when compared to keepalived-repmgr. Keepalived-repmgr also does not offer cross-containment and load balancing abilities. In Fig 4, the master database server CPU usage is high 80% most of the time while in Fig 16, HAProxy shares the read-loads so that master and other standbys’ CPU usages are close values. HAProxy-PgBouncer also provides two different methods to implement failovers: auto-failover and farm-failover.
In our future work, we intend to investigate further the failover analysis of the two clusters, at the present we observe a difference of the HAProxy-PgBouncer cluster - when the mater failure is detected by repmgrd, from the keepalived-repmgr cluster in which the failure is detected by keepalived.

ACKNOWLEDGMENT

Our deepest heartfelt thanks are given to Andrew Achtenberg, David Berry, Hong Zhang, Manish Rajani, Monir Iskarous, Sachin Agarwal and Shilpa Bhullar; and also the Organizers of PGConf US 2016.

REFERENCES


[26] RDF 1.1 Primer, http://www.w3.org/TR/rdf11-primer/


[31] SPARQL 1.1 Overview, http://www.w3.org/TR/sparql11-overview/


