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 computers 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 throughputs 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, repmgrd.

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

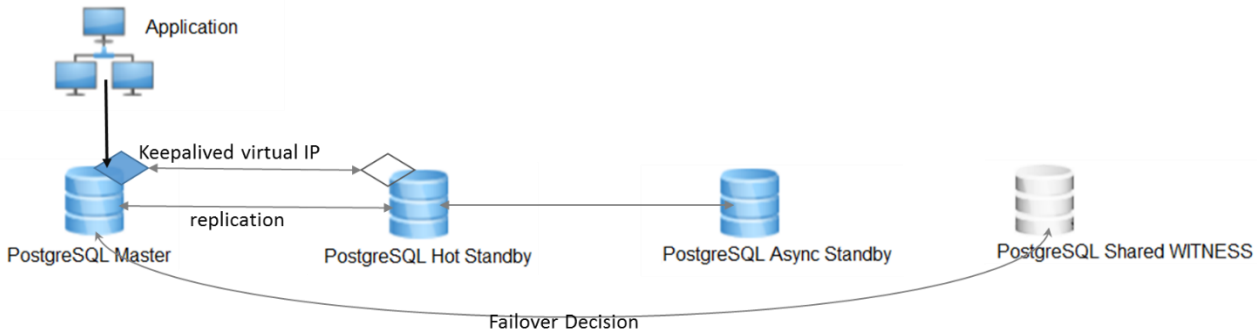

Fig. 1. Theoretical model of keepalived-repmgr cluster

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 back-end 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 Skounakis, 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 Heflin, 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 Narsimha, 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 Sarnin, 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.

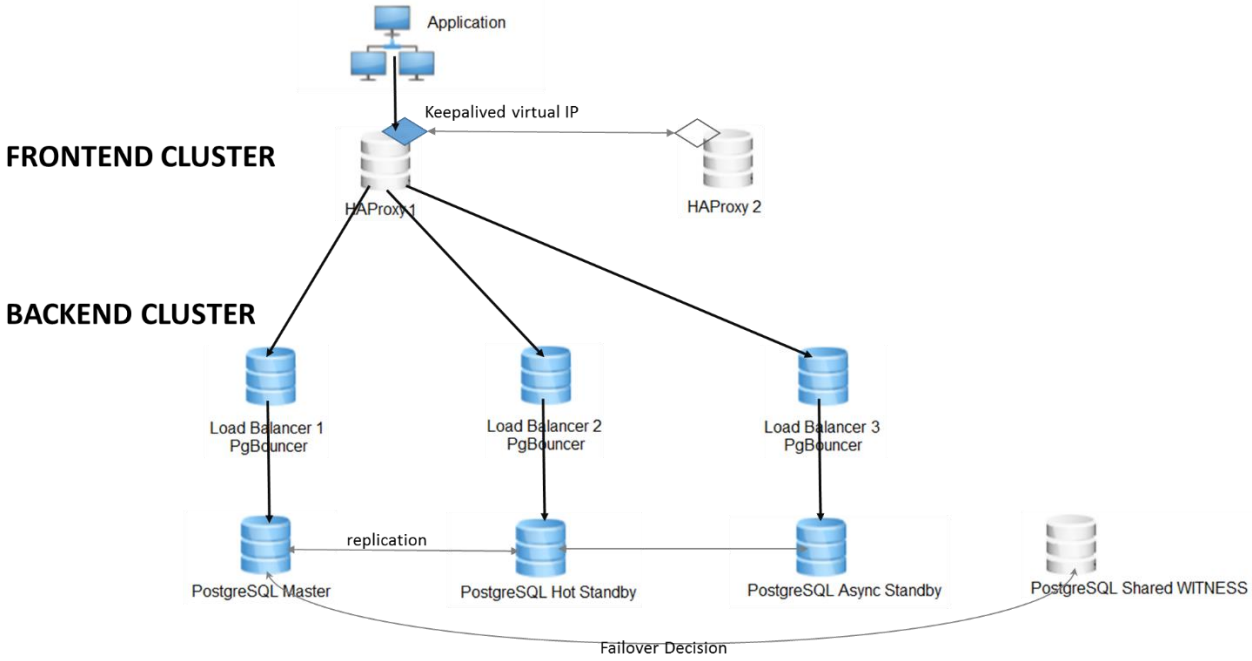

Fig. 2. Theoretical model of HAProxy-PgBouncer cluster engages to load balancing.

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 manger watch-dog) 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

$$Throughput = \frac{Number\ of\ Transactions}{Real\ Execution\ in\ seconds} \quad (1)$$

$$KB/\text{sec} = \frac{Throughput * Avg.Bytes}{1024} \quad (2)$$

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 I
KEEPALIVED-REPMGR / JMETER

HTTP Request	Test duration	Avg. Response Time /sample	Throughput	KB/sec	Avg. Bytes /transaction	Avg. Latency /transaction
Read Only without data execution	463.976s	27.494s	2,155.284	609.467	289.565	27.443
Read Only with data execution	478.775s	29.951s	2,088.664	8,063.858	3,953.432	29.909
Simple Write with Inserts and Updates	753.529s	58.480s	1,327.089	376.119	290.219	58.418
Simple Write with Deletes	533.122s	31.481s	1,875.743	530.419	289.565	31.421
Read Write with Selects, Inserts and Updates	981.059s	80.431s	1,019.307	288.666	289.995	80.372
Read Write with Selects and Deletes	570.773s	37.486s	1,752.010	493.719	288.565	37.426

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.

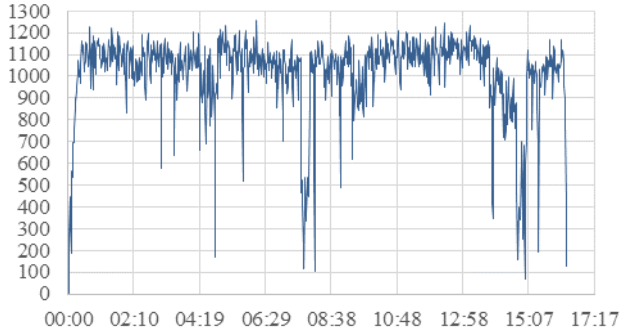

Fig. 3. Keepalived-repmgr: Read Write with Selects, Inserts and Updates - Transactions per Second

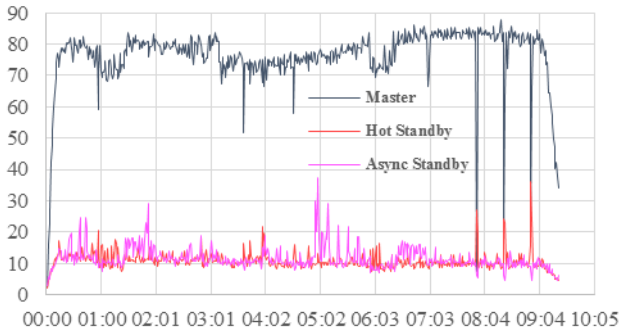

Fig. 4. Keepalived-repmgr: Read Write with Selects and Deletes - CPU Usages

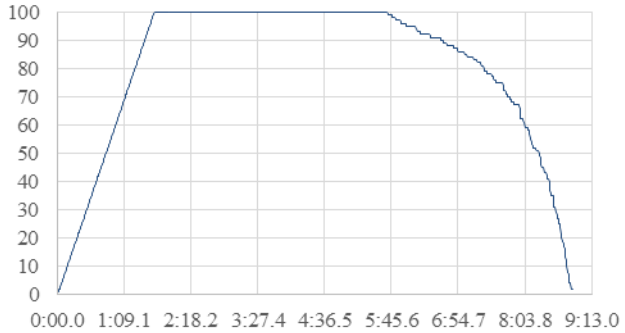

Fig. 5. Keepalived-repmgr: Simple Write with Deletes - Active Threads

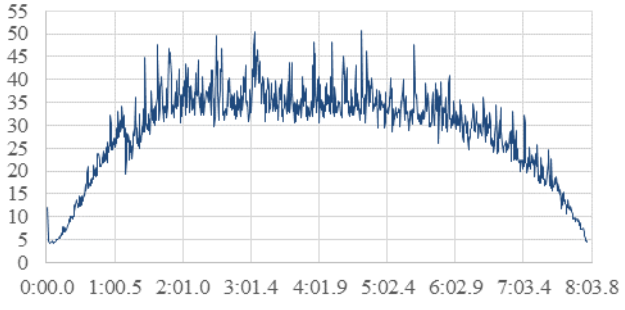

Fig. 6. Keepalived-repmgr: Read Only with Data Execution - Response Time

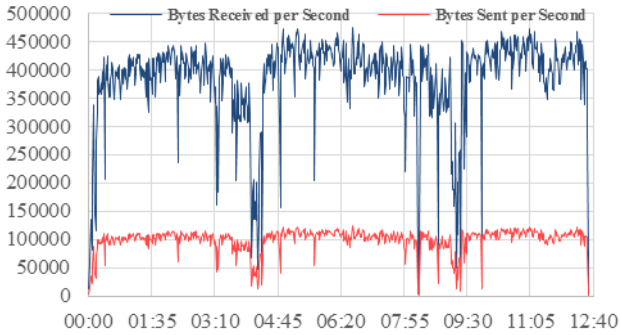

Fig. 7. Keepalived-repmgr: Simple Write with Inserts and Updates - Bytes Throughput over Time

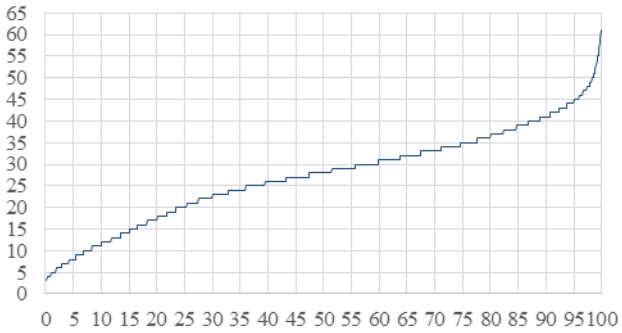

Fig. 8. Keepalived-repmgr: Read Only without Data Execution - Response Times Percentiles

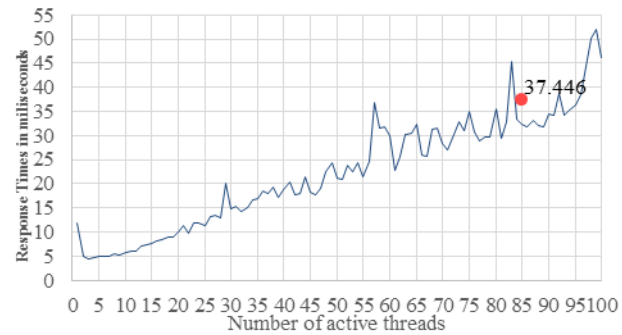

Fig. 9. Keepalived-repmgr: Read Write with Selects and Deletes - Response Times vs Threads

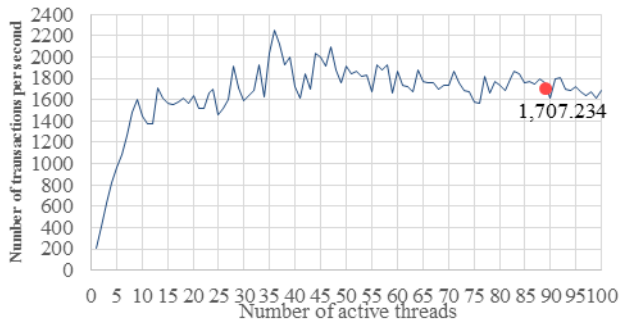

Fig. 10. Keepalived-repmgr: Simple Write with Inserts and Updates - Transaction Throughput vs Threads

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.

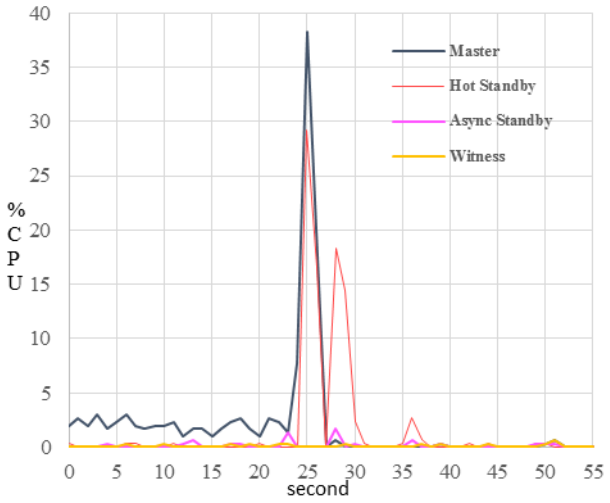

Fig. 11. Keepalived-repmgr: Failover CPU usages

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%.

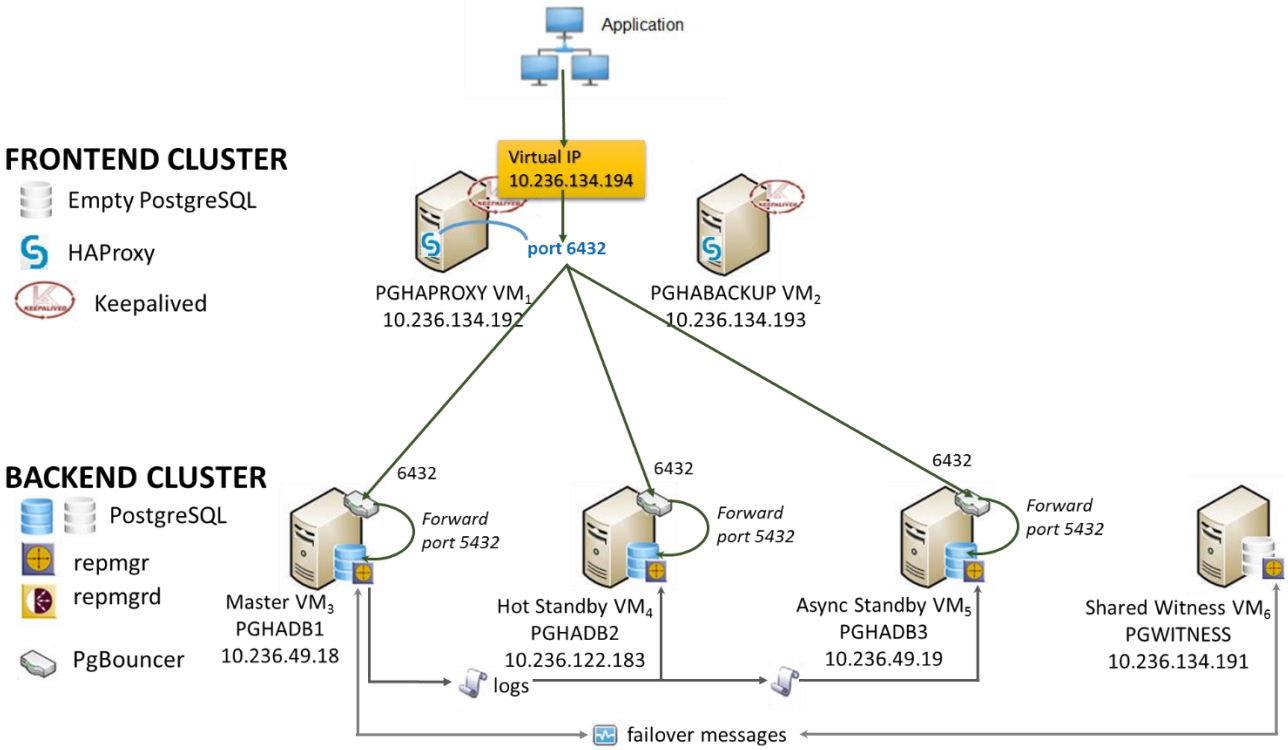

Fig. 12. Implementation of HAProxy-PgBouncer cluster

TABLE II
HAPROXY-PGBOUNCER / JMETER

HTTP Request	Test duration	Avg. Response Time /sample	Throughput	KB/sec	Avg. Bytes /transaction	Avg. Latency /transaction
Read Only without data execution	423.901s	28.234s	2,359.041	662.477	287.565	28.228
Read Only with data execution	471.192s	28.215s	2,122.277	1,354.034	653.322	28.209
Simple Write with Inserts and Updates	702.484s	55.893s	1,423.520	1,895.078	1,363.212	55.886
Simple Write with Deletes	521.546s	36.755s	1,917.376	540.319	288.565	36.749
Read Write with Selects, Inserts and Updates	970.949s	77.564s	1,029.920	679.687	675.780	77.557
Read Write with Selects and Deletes	568.803s	42.116s	1,758.078	626.928	365.157	42.110

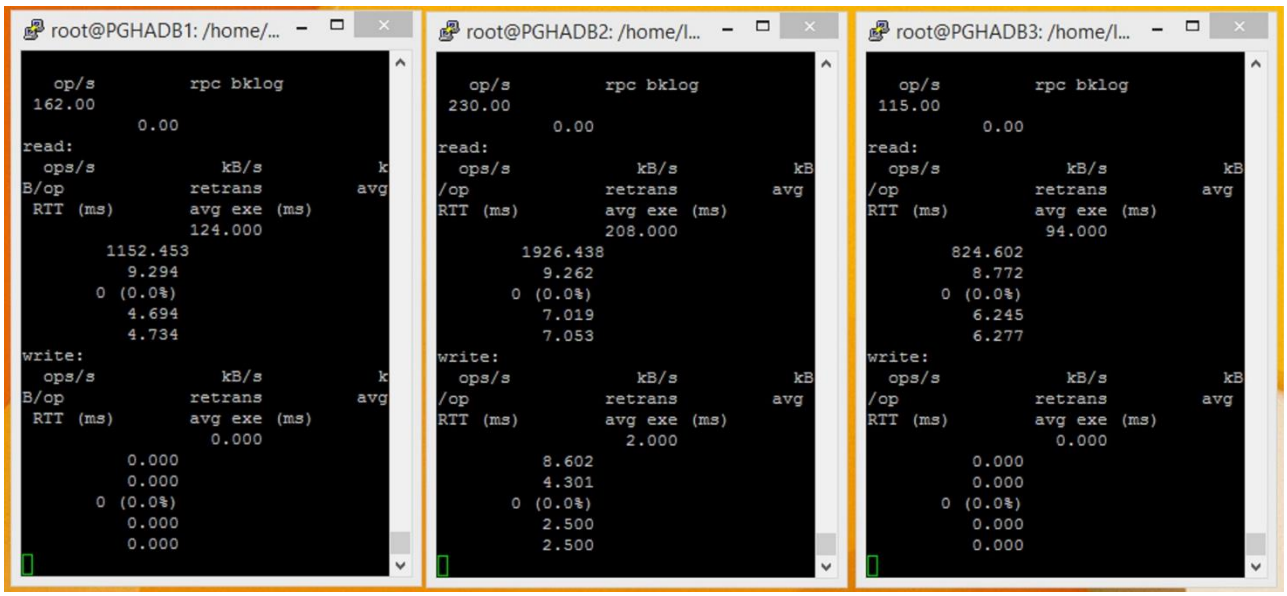

Fig. 13. Load balancing observed by sysstat on the master PGHADB1, the hot standby PGHADB2 and the async standby PGHADB3

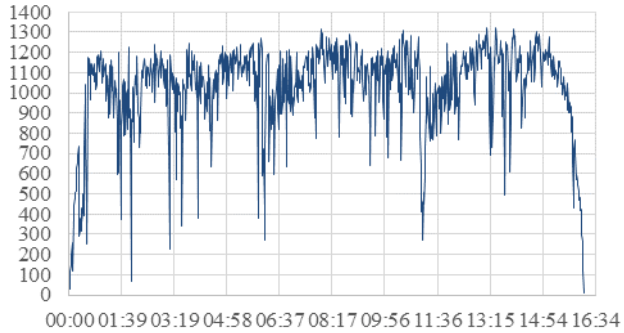

Fig. 14. HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Transactions per Second

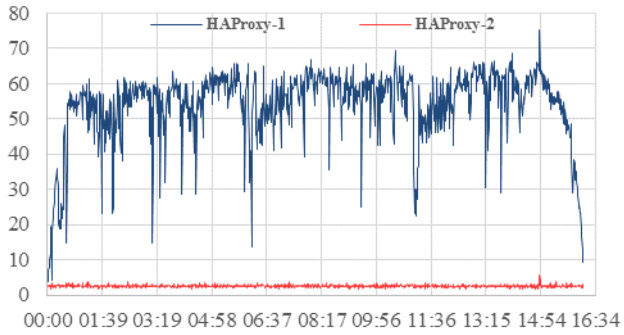

Fig. 15. HAProxy-PgBouncer: Read Write with Selects, Inserts and Updates - Frontend CPU usages

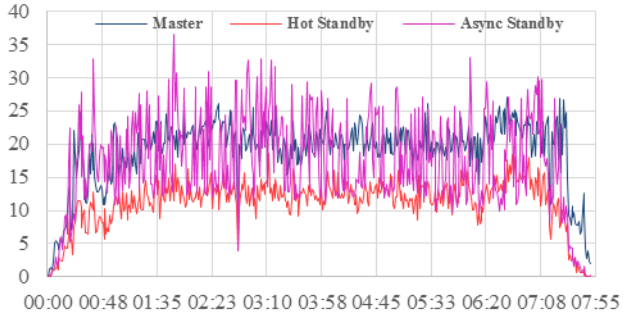

Fig. 16. HAProxy-PgBouncer: Read Only with Data Execution - Backend database server CPU usages on Load Balancing

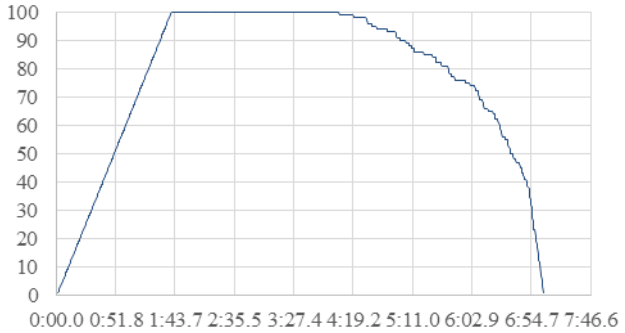

Fig. 17. HAProxy-PgBouncer: Read Only without Data Execution - Active Threads

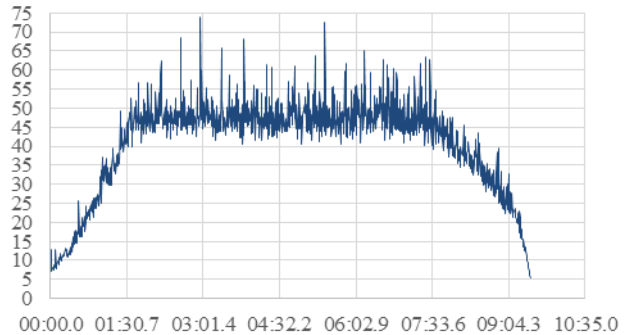

Fig. 18. HAProxy-PgBouncer: Read Write with Selects and Deletes - Response Time

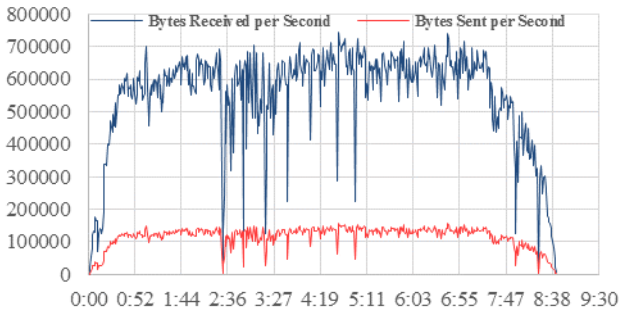

Fig. 19. HAProxy-PgBouncer: Simple Write with Deletes - Bytes Throughput over Time

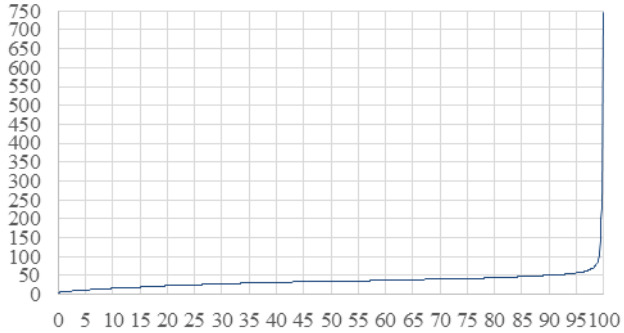

Fig. 20. HAProxy-PgBouncer: Simple Write with Deletes - Response Times Percentiles

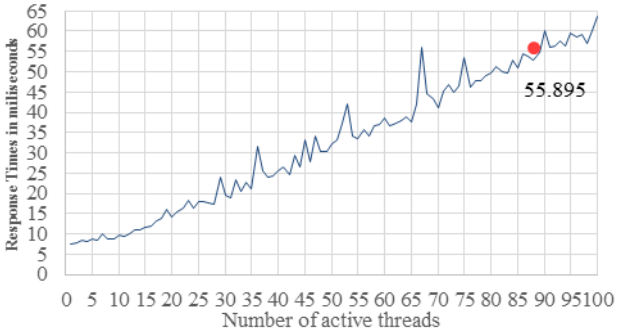

Fig. 21. HAProxy-PgBouncer: Simple Write with Inserts and Updates - Response Times vs Threads

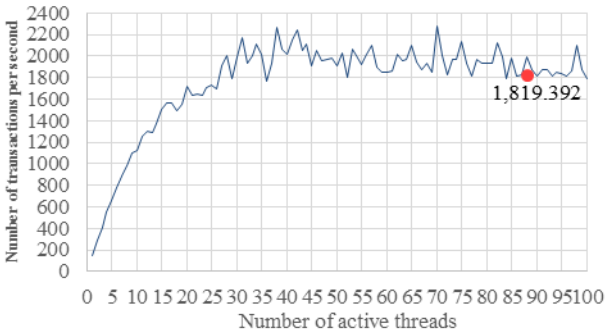

Fig. 22. HAProxy-PgBouncer: Simple Write with Inserts and Updates - Transaction Throughput vs Threads

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.

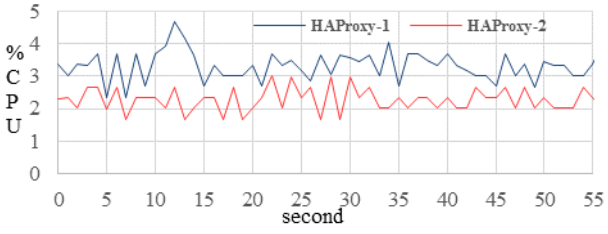

Fig. 23. HAProxy-PgBouncer: Failover Front-end CPU usages

Fig 23 and Fig 24 show the Front-end and Back-end CPU usages during a failover of HAProxy-PgBouncer.

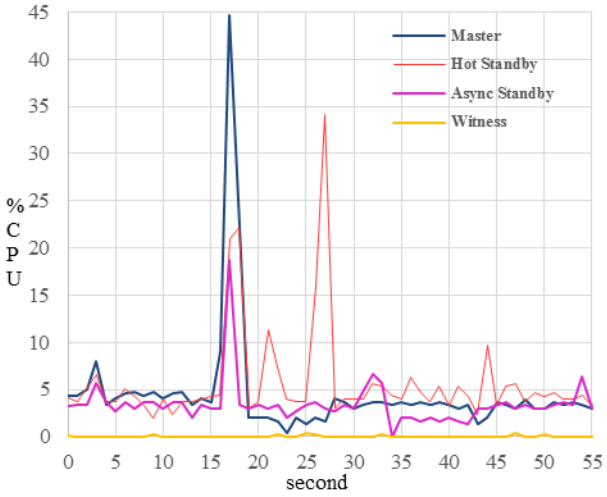

Fig. 24. HAProxy-PgBouncer: Failover Back-end CPU usages

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
THROUGHPUT IMPROVEMENTS OF HAPROXY-PGBOUNCER CLUSTER IN COMPARED TO KEEPALIVED-REPMGR CLUSTER

HTTP Request	Throughput improvement
Read Only without data execution	9.454%
Read Only with data execution	1.609%
Simple Write with Inserts and Updates	7.266%
Simple Write with Deletes	2.220%
Read Write with Selects, Inserts and Updates	1.041%
Read Write with Selects and Deletes	0.346%

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

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving relations for cache performance", the 27th International Conference on Very Large Data Bases, pp. 169-180, 2001.
[2] A. K. Dwivedi, C. S. Lamba, and S. Shukla, "Performance Analysis of Column Oriented Database versus Row Oriented Database", International Journal of Computer Applications (0975 – 8887), Vol. 50, No. 14, July 2012
[3] A. Saikia, S. Joy, D. Dolma, and R. Mary. R, "Comparative Performance Analysis of MySQL and SQL Server Relational Database Management Systems in Windows Environment", IJARCCE, Vol. 4, Iss. 3, pp. 160-164, Mar 2015.
[4] Apache Jena, https://jena.apache.org/
[5] BSBM V3.1 Results (April 2013), http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V7/index.html
[6] C. Bizer and A. Schultz, "The Berlin SPARQL Benchmark", International Journal on Semantic Web and Information Systems (IJSWIS), Vol. 5(2), pp. 1-24, 2009.
[7] D. Agrawal, S. Das, and A. E. Abbadi, "Big Data and Cloud Computing: Current State and Future Opportunities", in Proceedings of the EDBT 2011/ACM, Uppsala, Sweden, Mar 2011.
[8] D. J. Abadi, "Column Stores for Wide and Sparse Data", in CIDR, pp. 292-297, Asilomar, CA, USA, 2007.
[9] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach. "Scalable semantic web data management using vertical partitioning", the 33rd International Conference on VLDB, pp. 411-422, 2007.
[10] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, "Materialization Strategies in a Column-Oriented DBMS", in ICDE, 2007.
[11] D. J. Abadi, S. R. Madden, and M. C. Ferreira, "Integrating Compression and Execution in Column-Oriented Database Systems", in SIGMOD, pp. 671-682, 2006.
[12] Datastax Corporation, "Introduction to Apache Cassandra", white paper, San Mateo, Calif., available at datastax.com, Jul 2013.
[13] Datastax Corporation, "The Modern Online application for the Internet economy: 5 Key Requirements that Ensure Success", white paper, Santa Clara, Calif., available at datastax.com, 2014.
[14] DB-Engines.com, "DB-Engines Ranking of Relational DBMS", http://db-engines.com/en/ranking/relational+dbms, at Dec 2015.
[15] Fuseki, http://jena.apache.org/documentation/serving_data/index.html
[16] J. M. Emison, "2014 State of Database Technology", Information Week, San Francisco, CA, Rep. R7770314, 2014.
[17] Lee J., "Oracle vs. MySQL vs. SQL Server: A Comparison of Popular RDBMS", available: https://blog.udemy.com/oracle-vs-mysql-vs-sql-server/, Nov 2013.
[18] M. Horridge and S. Bechhofer, "The OWL API: A Java API for OWL ontologies", Semantic Web Journal, Vol. 2(1):11-21, 2011.
[19] M. N. Vora, "Hadoop-HBase for Large Scale Data", in Proceedings of the IEEE International Conference on Computer Science and Network Technology, pp. 601-605, Dec 2011.
[20] N. F. Naim, A. I. M. Yassin, W. M. A. W. Zamri, and S. S. Sarnin, "MySQL Database for Storage of Fingerprint Data", IEEE UKSim, Vol. 62, pp. 293-298, 2011.
[21] O. Erling and I. Mikhailov, "RDF Support in the Virtuoso DBMS", Networked Knowledge-Networked Media, Springer Berlin Heidelberg, pp. 7-24, 2009.
[22] P. A. Boncz and M. L. Kersten, "MIL primitives for querying a fragmented world", VLDB, Vol. 8, Iss. 2, pp. 101-119, Oct 1999.
[23] P. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-pipelining query execution", in CIDR, pp. 225-237, 2005.
[24] P. Orduña, A. Almeida, U. Aguilera, X. Laiseca, D. López-de-Ipiña, and A. G. Goiri, "Identifying Security Issues in the Semantic Web: Injection attacks in the Semantic Query Languages", JSWEB, Vol. 51, pp. 4529-4542, Valencia, Spain, Sep 2010.
[25] RAP - RDF API for PHP, http://wifo5-03.informatik.uni-mannheim.de/bizer/rdfapi/
[26] RDF 1.1 Primer, http://www.w3.org/TR/rdf11-primer/
[27] S. Abiteboul, R. Hull, and V Vianu, "Foundations of Databases: The Logical Level", Addison-Wesley Longman, 1995.
[28] S. Gupta and N. Narsimha, "Performance Evaluation of Nosql-Cassandra over Relational Data Store-MySQL for Bigdata". International Journal of Technology, Vol. 6(4), pp. 640-649, 2015.
[29] S. Khoshafian, G. Copeland, T. Jagodis, H. Boral, and P. Valduriez, "A query processing strategy for the decomposed storage model", in ICDE, pp. 636-643, 1987.
[30] S. Kulshrestha and S. Sachdeva, "Performance Comparison for Data Storage - Db4o and MySQL Databases", the 7th International Conference on Contemporary Computing (IC3), IEEE, pp. 166-170, ISBN: 978-1-4799-5172-7, Noida, 2014.
[31] SPARQL 1.1 Overview, http://www.w3.org/TR/sparql11-overview/
[32] T. Sandholm and D. Lee, "Notes on Cloud computing principles", Journal of Cloud Computing: Advances, Systems and Applications, Vol. 3(21), pp. 1-10
[33] The Welcome Project, available at http://www.welcome-project.eu
[34] V. Kilintzis, N. Beredimas, and I. Chouvarda, "Evaluation of the performance of open-source RDBMS and triplestores for storing medical data over a web service", the 36th Annual International Conference of the IEEE Engineering in Medicince and Biology Society, pp. 4499-502, 2014
[35] V. N. Gudivada, D. Rao, and V. V. Raghavan, "Nosql Systems for Bigdata Management", in Proceedings of the 10th World Congress on Services, IEEE, Vol. 42, pp.190-197, 2014.
[36] Y. Bassil, "A Comparative Study on the Performance of the Top DBMS Systems", Journal of Computer Science & Research, Vol. 1, No. 1, pp. 20-31, Feb 2012.
[37] Y. Guo, Z. Pan, and J. Heflin, "LUBM: A Benchmark for OWL Knowledge Base Systems", Web Semantics: Science, Services and Agents on the World Wide Web archive, Vol. 3(2-3):158-182, 2005.
[38] Z. Gansen, W. Huang, S. Liang, and Y. Tang, "Modelling MongoDB with Relational Model", in Proceedings of the Fourth International Conference on Emerging Intelligent Data and Web Technologies, IEEE, Vol. 25, pp. 115-121, 2013.

