HIGH AVAILABILITY AND LOAD BALANCING FOR POSTGRESQL DATABASES: DESIGNING AND IMPLEMENTING.

Pablo Bárbaro Martinez Pedroso

Universidad De Las Ciencias Informáticas, Cuba

ABSTRACT

The aim of the paper is to design and implement an approach that provides high availability and load balancing to PostgreSQL databases. Shared nothing architecture and replicated centralized middleware architecture were selected in order to achieve data replication and load balancing among database servers. Pgpool-II was used to implementing these architectures. Besides, taking advantage of pgpool-II as a framework, several scripts were developed in Bash for restoration of corrupted databases. In order to avoid single point of failure Linux HA (Heartbeat and Pacemaker) was used, whose responsibility is monitoring all processes involved in the whole solution. As a result applications can operate with a more reliable PostgreSQL database server, the most suitable situation is for applications with more load of reading statement (typically select) over writing statement (typically update). Also approach presented is only intended for Linux based Operating System.

KEYWORDS

Distributed Database, Parallel Database.

1. INTRODUCTION

With increasing and fast development of connectivity, web applications have growth enormously, in part because of the growing of the amount of clients and the requests of these as well. These applications work with an underlying database. Having interruptions of the database may cause unexpected functioning. For small applications this could be permissible, however for critical applications constant interruptions of databases are unacceptable. For instance, applications in airport that monitor and control air traffic should be equipped with a database capable of serving the data every time that is requested. If this database gets down this airport may suffer a serious catastrophe. There exits tons of phenomena that could crashes services provided for databases. Disconnections or failures of electronic devices responsible of guarantee connectivity such as wires, switches, routers, etc.; operating systems failures: main memory, processors, storage devices, short circuit, unexpected outage; software malfunction of Data Base Manager System, cyber-attacks, undetected and unsolved programming bugs; even natural phenomena such as cyclones, tornados, earth quakes, thunders and fires. Any of these situations could stop database servers from continuing offering services. For this reason is necessary to design and implement a solution that provides high availability and load balancing to databases.

High availability to databases can be accomplished throw replication of data, which is storing and maintaining of one or more copies or replicas of databases in multiples sites (servers, nodes)(Raghu Ramakrishnan 2002). This is useful to support failover, a technique that enables automatic redirection of transactions from a failed site[1] to another that stores a copy of the data(M. Tamer Özsu 2011).

A replication solution is based mainly in two decisions: where updates are performed (writing statements) and how propagate the updates to the copies (M. Tamer Özsu 2011). In (Jim Gray 1996) are defined two alternatives of where updates can be performed. The first called group (master-master, update anywhere) states that any site with a copy of data item can update it. And the second called master (master-slave) proposes that each object (data item) belongs to a master site. Only the master can update the primary copy of the object. All other replicas are read-only, and waits for the propagation of updates carried out in master site.

Also Gray et al. 1996 defines two ways of propagating updates: eager (synchronous) and lazy (asynchronous) replication. Eager replication applies every update to all copies as a global transaction. As a result either all sites commit the updates or none does it. This ensures that all sites have full copy of data all the time. On the other hand, lazy replication first commits updates in master server (site owner of primary copy of object), and later propagates the updates to others replicas as a separate transaction for every server. As a result part of the time not all sites have data updated.

Different combinations can be obtained as the table below shows.

Table 1Combination of where and how Updates can be Performed.

Eager (synchronous) master (master-slave)	Lazy (asynchronous) master (master-slave)
Eager (synchronous) group (multi-master, update anywhere)	Lazy (asynchronous) group (multi-master, update anywhere)

Each of these solutions has its advantages and drawbacks depending of its use. Therefore the configuration must be selected (depending on its characteristics) for the most suitable environment or situation. In (Bettina Kemme 2010; M. Tamer Özsu 2011) is available conceptual basis of database replication while (Salman Abdul Moiz 2011) offers a survey of database replication tools.

2. Material And Methods

The schema selected is synchronous multi-master replication. As it was previously stated in synchronous replication every site has a full copy of data. As consequence, only one site is needed to process reading statement. Therefore load balancing can be achieved, since reading transactions can be distributed and processed in parallel among servers. This avoids break downs of services for overload causes. Also decreases response time to client applications, due to parallel processing of reading statements. The main drawback of this schema is that global transaction involving all sites slows down connectivity speed between them. Therefore this approach is recommended for systems with large volume of reading statement instead of tons of update transactions. Other important feature can be observed in failover mechanism. In case of the master node fails (master-slave situation), the slave promoted as new master, does not require additional operations to recover last consistent state of failed master. Wasting of time only occurs in detecting master node has failed and promoting a slave as new master. In multi-master case the

[1]Database server that for some reason (software malfunction, network failure, inconsistent state reached) is not able of provides services.

behavior is the same. For critical scenario where $n - 1$ nodes have failed, the last one remaining online and consistent can guarantee by itself the services until failed nodes get back online. Synchronous replication in multi-master scheme proposes a protocol named ROWA [2](Read One Write All). This means transactions reach commit step if and only if all sites update the changes. As can be seen if any of the sites is down none of the updates can be achieved (David Taniar 2008). This protocol assures consistency among nodes over availability. Despite high consistency among nodes is an important feature it is possible continuing accepting updates and later restoring down sites, so high availability is maintained along the process. In order to overcome previous limitation was designed ROWA-A protocol (Read One Write All Available) which indicates that updates can be processed for all sites that are still online (David Taniar 2008). Although this approach may compromise consistency on the servers that were not part of transaction when they get back online, nevertheless down sites could be identified and restored, so inconsistency will be solved.

2.1. Distributed Database Architecture

Physical design is needed to collocate data across sites (Raghu Ramakrishnan 2002). The architecture of distributed database system is based in shared nothing architecture. Each node has its own processor unit (P), main memory (M), store device (SD) and Database Manager System, all running on independents operating systems interconnected throw network. Since corrupted servers can be detached instantaneously and remaining servers can resume operations, high availability can be achieved; besides, new nodes could be added easily, which leads to a scalable solution(M. Tamer Özsu 2011; Raghu Ramakrishnan 2002).The image below shows the architecture adopted.

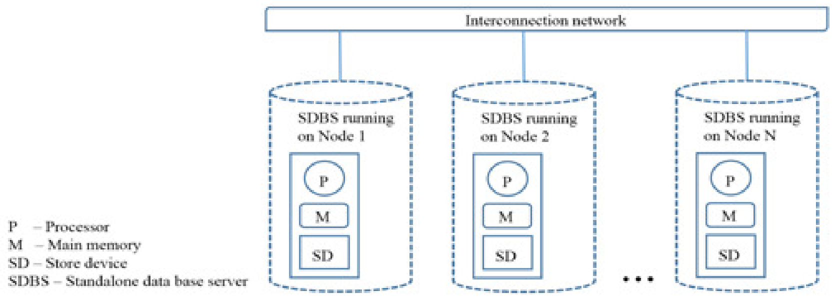

Figure 1:Shared Nothing Architecture. Bibliographic Source: (Raghu Ramakrishnan 2002).

2.2. Database Replication Architecture

The replication architecture is based on middleware, specifically replicated centralized middleware replication (Bettina Kemme 2010). This architecture removes single point of failure, since at least one stand by middleware is maintained waiting in case primary fails, in such case stand by middleware becomes primary and resumes operations. Middleware is placed between clients and server. It receives all client requests and forwards them to the relevant replicas. The resulting answers of the database servers are replied by the middleware to the corresponding clients. Writing statement (typically insert, update, delete) are send them to all sites, this allow to achieve full replication system. Because of this, reading statement (typically select) is processed only for one site. Load balancing can be achieved as high availability as well.

[2]In ROWA protocol, with at least one site down and one site online, data still can be served but it cannot be updated, and forward databases are considered unavailable.

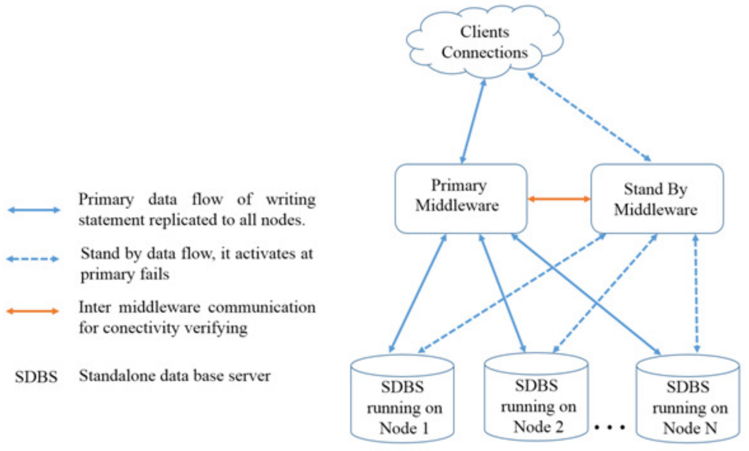
Clients Connections
Primary Middleware
Stand By Middleware
Primary data flow of writing statement replicated to all nodes.
Stand by data flow, it activates at primary fails
Inter middleware communication for conectivity verifying
SDBS Standalone data base server
SDBS running on Node 1
SDBS running on Node 2
SDBS running on Node N

Figure 2: Replicated Centralized Middleware Architecture. Bibliographic Source (Bettina Kemme 2010).

2.3. TOOLS

Taking into account these architectures some tools are needed for implementing them. This section will discuss such topic. Pgpool-II is used as PostgreSQL database middleware. It provides connection pooling[3], database replication and load balancing. Besides, pgpool-II implements ROWA-A protocol, thus when failover it occurs online database servers resume operations of the applications. This is an important feature because assures availability of database services even when at most $n - 1$ servers have failed, with n total number of database servers. As can be seen as more database servers are used more high is the level of availability of database services. At the same time, as more middle wares (instances of pgpool-II running on different sites) are used, more is the certainty that services still will be running after primary (or any stand by promoted as primary) fails. In order to provide high availability to Pgpool-II, PostgreSQL processes and network services was used Linux HA (High Availability). Linux HA is the origin of two projects Heartbeat and Pacemaker. Heartbeat provides messaging layer between all nodes involved while Pacemaker is a crm (cluster resource manager), whose purpose is to perform complex configurable operations to provide high availability to resources (processes, daemons, services) (Team 2015).This involves restart a stopped resource and maintain a set of rules or restrictions between them.

For a detailed description about how to install and configure these tools consider consult (Group 2014)in the case of Pgpool-II, (Project 2014)for Heartbeat and (Team 2015) for Pacemaker.

2.4 ARCHITECTURE OF THE APPROACH.

After previous decisions were made a more general architecture is presented (Figure 3). As can be seen, replicated centralized middleware is encapsulated at middle wares level whereas shared nothing architecture is at database servers level. At a higher level monitoring of processes is carried out at middle wares level and at database servers level as well throw Linux HA.

[3]"*Pgpool-II maintains established connections to the PostgreSQL servers, and reuses them whenever a new connection with the same properties (i.e. user name, database, protocol version) comes in. It reduces the connection overhead, and improves system's overall throughput*." Bibliographic source: GROUP, P. G. D. Pgpool-II Manual Page. In., 2014, vol. 2015.

Connections are represented by arrows, and should be faster enough in order to avoid a bottleneck at messaging and communication process.

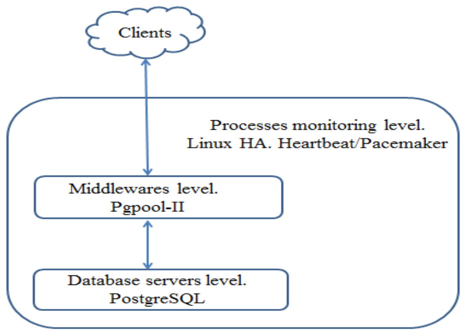

Figure 3: General architecture.

An interesting feature that presents this architecture is that is no restricted only for the tools used in this paper. Therefore different tools can implement this architecture, leading to be operating system independent as well.

3. Related Work

Previous works have implemented replicated centralized middleware architecture (Felix Noel Abelardo Santana and Méndez Roldán ; Partio 2007; Pupo 2013; Rodríguez 2011; Santana 2010) throw pgpool-II. Some of them use asynchronous master-slave scheme (Felix Noel Abelardo Santana and Méndez Roldán ; Partio 2007), while others are based on synchronous multi-master replication scheme (Pupo 2013; Rodríguez 2011; Santana 2010).

However none of these solutions has planned the restoration of a corrupted database. This is known as online recovery: the capability of restoring a crashed database while online databases still servicing clients. Also the monitoring of PostgreSQL instances had not been designed. Pgpool-II provides a two phase mechanism to achieve database restoration. First phase is triggered when a database is detected failed executing a script that performs a PostgreSQL physical backup from an online master to failed site. During this phase available sites process client's requests saving changes into WAL (Write Ahead Logs) files. Once physical backup has done first stage is finished. At starting second stage clients are disconnected, a second scripts is executed, which synchronizes WAL files generated during first phase to failed site, assuring that this site is fully recovered. After this the node is attached to the system and operations of databases are resumed. Taking advantage of Linux, scripts triggered during online recovery were developed in Bash, which is a native interpreter of Linux based Operating System, so no extra software is needed. Restoration of databases are based on Point in Time Recovery feature of PostgreSQL, pgpool-II as event trigger mechanism and Bash as executor of predefined routines encapsulated in multiples scripts. Recipe adopted for implementing this process was extracted from(Simon Riggs 2010) and follows the steps described in Algorithm 1.

Algorithm 1. General recovering process based on Point in Time Recovery.

```
1. CHECKPOINT
2. Starting of 1st Stage of Recovering. Execution of script copy_base_backup.
3. Waiting for clients disconnections. Depending on parameter setting client_idle_limit_in_recovery of configuration file pgpool.conf. Setting adopted is 0, which means waiting for clients disconnections by themselves, in order to avoid data loss on current transactions.
4. CHECKPOINT
5. Starting of 2ndStage of Recovering. Execution of script pgpool_recovery_pitr.
6. Starting of PostgreSQL process in node fully recovered. Execution of script pgpool_remote_start.
7. Attaching of node fully recovered.
8. Resuming of client connections.
```

In order to detail the whole process of recovering, Algorithms 2 and 3 present the procedures implemented in the scripts.

Let *failed node* database server crashed database server targeted to be recovered, *master node* database server from recovery process will be carried out and therefore scripts executed, *PGDATA* PostgreSQL database directory, *Backup-mm-dd-yyyy.tar.gz* file compressed with PGDATA PostgreSQL database directory with format date mm-dd-yyyy for month, day and year respectively.

Algorithm 2. Procedure encapsulated in script copy_base_backup.

```
1. Checking for connectivity to failed node. If false, avoid automatic online recovery and generate alert for later manual recovery. Otherwise, continue with automatic online recovery.
2. Stopping of PostgreSQL process on failed node throw Pacemaker.
3. Starting of archiving any transaction during 1stStage of recovering in WAL files.
4. Starting of physical backup carried out by PostgreSQL at master node.
5. Compressing of PGDATA directory to Backup-mm-dd-yyyy.tar.gz file.
6. Stopping of physical backup
7. Copying Backup-mm-dd-yyyy.tar.gzfile to failed node.
8. Decompressing Backup-mm-dd-yyyy.tar.gzto PGDATA directory in failed node.
```

Once 1st Stage is accomplished and there is not active client connections, routine programmed in Algorithm 3 is executed at 2nd Stage of recovering process. Its responsibility is synchronize changes made during 1st Stage (e.g. Algorithm 2). At the end of Algorithm 3, pgpool_remote_start script is triggered. Its function is the initialization of postgresql resource (e.g. PostgreSQL process) in order to Pacemaker can monitor such process. After this, recovering process is accomplished, failed node is fully recovered and clients can resume connections to middleware level.

Algorithm 3. Procedure encapsulated in script pgpool_recovery_pitr

```
1. Synchronization of WAL files generated during 1stStage, to failed node.
2. Stopping WAL archiving in master node.
```

PostgreSQL servers is where bash scripts are executed throw stored procedures invoked from pgpool-II. Because of this, pgpool-IIneeds to be set with recovery_user and recovery_password at pgpool.conf configuration file in order to connect to databases and trigger recovery stored procedure. Also it must provide as arguments the paths of bash scripts relative to PGDATA directory (e.g. location where bash scripts are placed in PostgreSQL database servers). These are recovery_1st_stage_command and recovery_2nd_stage_commandfor copy_base_backup and pgpool_recovery_pitr respectively. The script pgpool_remote_start does not need to be set, only be located in PGDATA directory. For creating recovery stored procedures please consult (Group 2014).

Online recovery can be done manually allowing not only restoration of a databases server, but adding new servers since it carries out a full recovery. Moreover online recovery is designed to be executed automatically in order to avoid database manager intervention.

Also none of the works surveyed that uses Heartbeat make use of Pacemaker (Felix Noel Abelardo Santana and Méndez Roldán; Pupo 2013; Rodríguez 2011; Santana 2010). Although Heartbeat contains a crm capable of giving high availability, this is a primitive version of pacemaker which is incapable of making complex operations as restriction of location and co location concerned to resources. Without this it is not possible guarantee high availability to PostgreSQL process and network services to all sites. Moreover, pgpool-II could be running and expecting client connections in one server while database services ip address is expecting database client connections on other server which could lead to unavailability of database services. To overcome this limitation co location restriction throw Pacemaker can be used.

As was stated previously processes monitoring will be carried out across the system throw Heartbeat/Pacemaker, therefore it needs to be installed on all server used.LSB class process init scripts pgpool2, postgresql and networking must be added as resources in order to monitor pgpool-II and PostgreSQL processes and networking services. OCF class script IPaddr2 must be also added as resource and as argument, must be set IP address (or subnet address). Once IPaddr2 is running will publish this address in order clients can connect with middleware level.

As can be seen some constraints must be defined in order to control resources behavior. Pgpool2 and IPaddr2 can only be executed on middleware level whereas postgresql on database server level. This is known as location constraints. On the other hand Pgpool2 and IPaddr2 should be running on the same middleware (e.g. the one that is active), otherwise pgpool-II could be running and expecting client connections in one middleware while IPaddr2 publishes database services ip address on other middleware which could lead to unavailability of database services.

3.1. Results And Brief Discussion

Shared nothing was selected as distributed database architecture which provides high availability and scalability; whereas replicated centralized middleware is used with synchronous multi-master replication, in order to keep databases servers fully synchronized. Some tools were used to accomplish all this. Pgpool-II was used as middleware responsible of replication of data, load balancer and framework for online recovery. On the other hand Heartbeat and Pacemaker were used for monitoring all processes involved in the solution such as pgpool-II, PostgreSQL instances, IPaddr2 (e.g. database services ip address), and restrictions between all these processes. Multiples scripts were developed in Bash in order to achieve the restoration of corrupted databases. Such restoration can be done manually and/or automatically, depending on requirements specified.

4. Conclusions

1. As a result of this investigation was designed and implemented a solution to provide high availability and load balancing to PostgreSQL databases. Therefore applications can count with a more reliable PostgreSQL database service.
2. Having multiples sites with full copy of data allows not only failover mechanism, but also, load balancing among the servers. Since every server has an entire and consistent copy of data, each of them can process all kind of reading statement. This situation avoids a breakdown of the services for overload causes.
3. Using more than one middleware (one active, at least one stand by) removes single point of failure, thus high availability can be obtained at middleware level.
4. Since distributed database architecture is shared nothing, many servers can be added or removed in a few configuration steps, because of this approach presented is flexible and scalable.
5. All tools selected are open source and developed for Linux, thus solution of this investigation is intended only for Linux based Operating System.

6. Since global transactions (of writing statement)slow down connectivity speed, this approach is more suitable and recommended for having big amounts of reading statement instead writing statement.

5. FUTURE WORK

1. Designing and carrying out several of experiments in order to measuring performance of heavy reading statements. Moreover, obtain a mathematical model which describes the phenomena of performance depending on number of database servers. This model could predict performance in a manner that for scalable solutions more precise decisions can be made.
2. Designing and implementing a solution intended for big amount of writing statement.Due to the fact that no solution fits all kind of problems, heavy write statement should be also explored in order to produce an approach in such sense.
3. Approach presented in this paper suffers from the fact that it is only intended for Linux based Operating Systems thus, is needed to extend this solution to other Operating Systems.

REFERENCES

[1] Bettina Kemme, R. J. P. A. M. P.-M. Database Replication. Edited By M.T. Özsu.Edtion Ed.: Morgan & Claypool, 2010. Isbn 9781608453825.

[2] David Taniar, C. H. C. L., Wenny Rahayu, Sushant Goel High-Performance Parallel Database Processing And Grid Databases. Edited By A. Zamoya. Edtion Ed. New Jersey: Wiley, 2008. 574 P.

[3] Felix Noel Abelardo Santana, P. Y. P. P., Ernesto Ahmed Mederos, Javier Menéndez Rizo, Isuel And H. P. P. Méndez Roldán Databases Cluster Postgresql 9.0 High Performance, 7.

[4] Group, P. G. D. Pgpool-Ii Manual Page. In., 2014, Vol. 2015.Http://Www.Pgpool.Net/Docs/Latest/Pgpool-En.Html

[5] Jim Gray, P. H., Patrick O'neil, Dennis Sasha. The Dangers Of Replication And A Solution. In Proceeding Acm Sigmod Int. Conference On Management Of Data. Monreal, Canada: Acm, 1996, P. 10.

[6] M. Tamer Özsu, P. V. Principles Of Distributed Database Systems. Edtion Ed.: Springer, 2011. 866 P. Isbn 978-1-4419-8833-1.

[7] Partio, M. Ev Aluation Of Postgresql Replica Tion And Load Balancingimplement A Tions. 2007.

[8] Project, L. H. Linux Ha User's Guide. In., 2014, Vol. 2015. Http://Www.Linux-Ha.Org/Doc/Users-Guide/Users-Guide.Html

[9] Pupo, I. R. Base De Datos De Alta Disponibilidad Para La Plataforma Videoweb 2.0. University Of Informatics Science, 2013.

[10] Raghu Ramakrishnan, J. G. Database Management Systems. Edtion Ed.: Mcgraw-Hill, 2002. 931 P. Isbn 0-07-246535-2.

[11] Rodríguez, M. R. R. Solución De Clúster De Bases De Datos Para Sistemas Que Gestionan Grandes Volúmenes De Datos Utilizando Postgresql. Universidad De Las Ciencias Informáticas, 2011.

[12] Salman Abdul Moiz, S. P., Venkataswamy G, Supriya N. Pal Database Replication: A Survey Of Open Source And Commercial Tools. International Journal Of Computer Applications, 2011, 13(6), 8.

[13] Santana, Y. O. Clúster De Altas Prestaciones Para Medianas Y Pequeñas Bases De Datos Que Utilizan A Postgresql Como Sistema De Gestión De Bases De Datos. Universidad De Las Ciencias Informáticas, 2010.

[14] Simon Riggs, H. K. Postgresql 9 Administration Cookbook. Edtion Ed. Birmingham, Uk: Packt Publishing Ltd., 2010. Isbn 978-1-849510-28-8.

[15] Team, P. D. Pacemaker Main Page. In., 2015, Vol. 2015.Http://Clusterlabs.Org/Wiki/Main_Page