OPEN ACCESS

International
Journal
Of Modern Engineering Research (IJMER)

To Study Effectiveness of Various Redundancy Architectures of Postgre SQL in Order to Achieve High Availability of Website

Ria Bhatt[1], Sumit Pandya[2]
[1]*Aravali Institute of Technical Studies Rajasthan Technical University, Kota*
[2]*School of Doctoral Research And Innovation GLS University, Ahmedabad*

***ABSTRACT**: Postgre SQL is a database management system, used in many different applications throughout the industry. As databases often are the bottlenecks in the performance of applications, their performance becomes crucial. Better performance can either be achieved by using more and faster hardware, or by making the software more efficient. In recent years the number of open-source database systems offering high availability functionality has exploded. The functionality offered ranges from simple one-to-one asynchronous replication to self-managing clustering that both partitions and replicates data automatically.*
This thesis is an attempt to evaluated database systems for use as the basis for high availability of a command and control system that should remain available to operators even upon loss of a whole datacenter.

I. INTRODUCTION

This chapter describes the main purpose and the problem statement that motivated and guided this thesis project.

1.1 Overview

Modern web development is challenged by new methodologies on all levels and it is demanding to be able to apprehend how new approaches connect together to deliver more reliable solutions and bring profit to its creators. For a background developer biggest revolution lies in creating redundant architecture of database to achieve high availability.

High availability is a feature which provides redundancy and fault tolerance. High availability computing and business critical services are two terms that traditionally involve significant cost and effort. Keeping a database available at all times can be a rough challenge, and for certain environments, redundancy is a must. Redundant data is not a backup, but can give you a failover in case your primary method of accessing data becomes unavailable. Availability is degree to which a system is up and running.

A typical scenario is that each high availability service is hosted by two x86 servers, where one of these is inactive and serves as a backup server in case of a failure on the active one. Due to business critical data hosted on these high availability servers, separation is often required. This means that each pair of servers can only host one service, or a group of services that share the same business critical data (see figure 1.1). The average server utilization today is at 25% according to Bittman and Scott in Gartner1 [1].

1.2 Motivation

The motivation is to describe how all those new modern approaches should be design to work together to take advantage from each other. PostgreSQL has built-in functionality for High Availability like Warm Standby, Hot Standby and Streaming Replication, missing few features like Switchover/Switchback, failover automation, minimal downtime etc. This work presents system level techniques that capitalize on the memory access patterns of database servers and the transactional properties of a DBMS to provide low overhead and cost effective high availability to commodity database systems.

Fig. 1.1 The traditional way of hosting business critical high available services, recognized by separation of services on different physical computers and redundant set of physical computers for each service. 10 services require 20 computers**.**

1.3 Problem statement

In modern application development, database servers play a crucial part in the overall performance. In principle an application and its underlying infrastructure should adapt to the dynamically changing conditions (demands and available resources at various costs) to promote the availability and reliability of a service, while minimizing the cost for the application service provider. With the increase in numbers and size of on-line communities there has been an increasing effort to exploit cross-functionalities across these communities. This dynamic nature of demand and traffic drives the need for a massively scalable solution to enable the availability (and reliability) of web-based applications.

Two main questions we wanted to answer with this thesis are the following:

- What is importance of redundant database architecture in public domain software engineering?
- What is the major role of redundant database in improving software engineering industry of India?

1.4 Aim and Objective

The thesis analysis available technologies and methodologies to achieve high availability of web application. Our major concern is reliability and availability that focuses on guaranteeing accessibility of system in difficult condition.

1.5 Terminology

1.5.1 PostgreSQL Introduction

PostgreSQL is an object-relational database management system and runs on all major operating systems. It implements the majority of the SQL:2011 standard [2] and it is ACID-compliant. The transaction model is based on multi-version concurrency control to avoid locking issues and dirty reads.

The history of PostgreSQL started in the late 1970's when its ancestor, Ingres, began as a project at the University of California, Berkeley. It has since then evolved and became PostgreSQL in 1996 when support for SQL was added. Today, PostgreSQL is an open source project that is maintained by the PostgreSQL Global Development Group, an international group of companies and individuals.

1.5.2 High availability

High availability computing is, as the term implies, a highly available computer system. How available the system should be depends on what the system should provide. Depending on the service, an outage of 1 second might be insignificant or disastrous. Therefore, the degree of availability should match the purpose and suit the business needs of the company.

A service level agreement (SLA) is a formal document of the promise made by the service provider to the customer about service provision policy [3, 4]. It is important that planned and unplanned outages do not exceed this service level degree [5].

If the electricity on one power source is lost, the servers will be unaffected and still available. By making the dependencies redundant, we end up with one single physical computer that is completely redundant with no single points of failure - the result is a fault tolerant computer.

1.5.3 Reliability

While availability is usually defined in terms of probability that the system is operating correctly at a point in time during continuous operation, reliability is defined as the probability that the system keeps operating correctly without failures for a defined period of time.

II. LITERATURE SURVEY

Database servers can work together to allow a second server to take over quickly if the primary server fails (high availability), or to allow several computers to serve the same data (load balancing). But keeping two machines synchronized is not easy task. Solutions deal with synchronization by allowing only one server to modify the data. Servers that can modify data are called read/write or "master" servers. Servers that can reply to read-only queries are called "slave" servers. Servers that cannot be accessed until they are changed to master servers are called "standby" servers. No data is lost.

In contrast, asynchronous solutions allow some delay between the time of a commit and its propagation to the other servers, opening the possibility that some transactions might be lost in the switch to a backup server,

and that load balanced servers might return slightly stale results. Asynchronous communication is used when synchronous would be too slow.

High Availability Clusters are often used by websites serving 24x7x365 not affording any downtime Eg: Amazon.com,Music websites, Customer Service sites etc., or Companies with Critical Databases

2.1 Significance

High Availability provides system to mitigate under disaster condition. Thus to maintain data or to avoid loss of data redundant copies of data are kept. It is important to reduce the down time, by providing continuous services when system components fail. This can be achieved by clustering.

2.2 Measuring availability

The degree of availability is an important factor in high availability systems. To be able to formalize the degree of availability in the service level agreement, we need a numerical value that represents our demand for the degree of availability. Hence, the service provider needs a method for calculating the availability in the active systems.

Based on historical data, we can calculate how available the system has been, the last week, month or year for example. This can be used as an expectation value of how available we expect the system to be in the future [5]. Knowledge about how often failures occur (MTBF) and the time to repair the failures (MTTR) can be used in this formula for calculating availability. [6, 7]

Availability =MTBF / (MTBF +MTTR)

The availability of the system is dependent on the failure rate of the equipments. These failure rates are often measured in the mean time before failure (MTBF) for each of the components in the system.

2.3 Building Highly Available Database Systems

This section is describing about hardware solutions for high availability. We then present techniques that have been used to implement highly available database systems.

2.3.1 High Availability through Hardware

Hardware reliability is an important part of implementing a highly available solution. Today, systems are built in a modular fashion where each hardware module can fail and be repaired or replaced independently of other modules in the system. Hardware solutions for achieving high availability typically involve redundancy of individual modules, or of the entire system (through clustering).

Individual modules, in turn, may have also been designed with internal redundancy. In this case, the module will fail only if a majority of the internal (redundant) components fail.

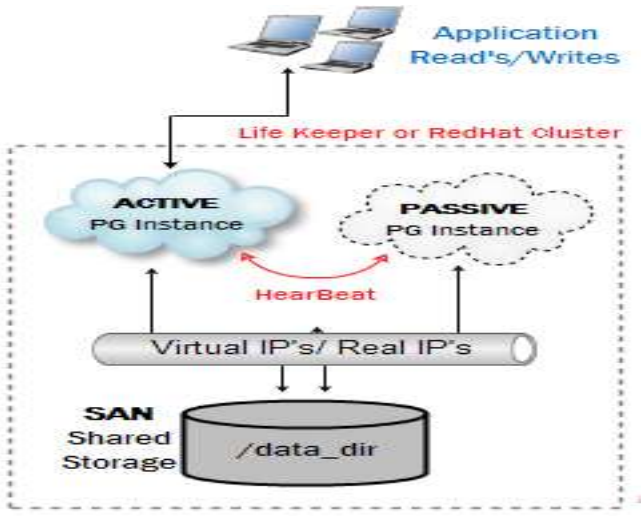

Fig. 2.1. High Availability Clustering with PostgreSQL

The cluster can have one of the following three configurations:

1. Active/cold configuration where one of the servers is active while the second is a cold backup, i.e., not serving workloads or attempting to keep up with the active server.
2. Active/passive configuration where one of the servers is active while the second is running the same workload (with some lag) and thus is a warm standby or log shipping.

3. Active/active configuration where both the servers are serving workloads actively. (is not supported by PostgreSQL.)

The primary and secondary servers may be geographically distributed. This provides better isolation from faults that may affect one site but not both sites at the same time.

In a standby database configuration, a production database ships its transaction logs to a remote site where the transactions are applied to a database running in recovery mode. The primary database runs in archive log mode. On a defined schedule, the transaction Write Ahead Logs (WAL) are copied to the remote location and applied to the standby database. Recovery mode simply means that the standby database is closed to user access so the WAL files can be applied. Once the log files are applied, a standby database is completely synchronized with the production database and can be brought online in the case of a failure.

Heartbeat is another open source solution to high availability clustering. The basic idea of Heartbeat is that the nodes in the cluster broadcast their status as heartbeats with information about which services are running on that node. This way, the other nodes in the cluster are able to know which nodes and services which are up and available and which are not. It is developed by the High-Availability Linux project. A heartbeat is a sensing mechanism which sends a signal across to the primary node, and if the primary node stops responding to the heartbeat for a predefined amount of time, then a failover occurs automatically [8].

2.3.2 High Availability for Database

Hardware solutions for high availability only provide limited advantage when faced with faults in software applications. Applications, such as database systems, couple hardware solutions with software techniques to implement a highly available system.

Two common approaches for providing high availability for database systems:-

Database Replication

Replication is a technique where a database is fully or partially replicated locally or at a remote site to improve performance and data availability [9]. The original copy of the data is referred to as the primary (or master) replica while the replicated copy is called a secondary replica. If the primary replica becomes unavailable, the data still remains accessible through the secondary replica. We can also have N-way replication where a primary replica is copied to N-1 other replicas. However, this additional replication comes at a higher cost. Replication may be implemented within a single database system or between database systems across machines possibly distributed across geographical boundaries. Changes from the primary replica are periodically propagated to the secondary replica. Synchronous or asynchronous replication is the two options for keeping the secondary replica up-to-date with the primary replica.

Synchronous replication [10, 11] is usually implemented through a read any, write all mechanism. Where a read transaction can read data from any replica but a write transaction must update all replicas. For this type of replication, the updating transaction must acquire exclusive access to all the replicas which might involve lock requests across remote sites. Also, in order for an update transaction to successfully commit, all the replicas must remain accessible during the transaction. If the replicas are distributed among remote and local sites, a two-phase commit protocol is required to ensure that each transaction either commits at all replicas or is aborted. Synchronous replication offers the advantage of keeping all the replicas strongly consistent. However, due to the high operational costs associated with acquiring locks, and excessive message passing for two-phase commit, it is rarely used in practice [10].

On the other hand, asynchronous replication [10] propagates the changes from the primary to the secondary through transaction log shipping or snapshots. The changes in the log records or a snapshot are then applied to the secondary copy. Asynchronous replication offers a trade-off in terms of minimizing overhead of normal operation and data consistency. With asynchronous replication, it is possible for a transaction to get slightly different results when accessing different replicas because they are updated only periodically. In a typical setting, asynchronous replication is run with two servers, a primary or a master, and a secondary, with the secondary server's database state being transitionally consistent to that of primary server with some replication lag. Also, typically, only the primary server can update the replicated data (e.g., in primary site asynchronous replication). When the primary server fails, the secondary server takes over execution losing some work, for example, in-flight transactions are aborted and restarted on the secondary server. Also, the secondary server needs to be brought up-to-date after a failure, i.e., the transaction log of committed operations performed on the primary server that were not yet propagated to the secondary server needs to be replayed at the secondary. After a failure, the primary server can be recovered while the backup server acts as the primary. Later, the roles of the two serves can be switched again, or the backup server can remain as the primary, and vice versa.

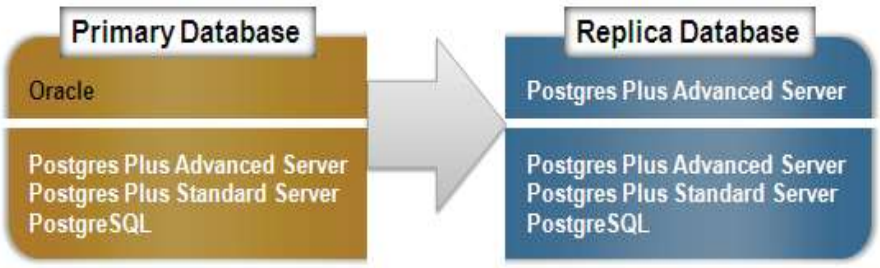

Fig. 2.2 Postgres Replication Options

Parallel Database Systems

Parallel database systems [12], typically running on clusters of physical machines, are a popular choice for implementing high availability for database systems. A parallel (or clustered) database system executes database operations (e.g., queries) in parallel across a collection of physical machines or nodes. Each node runs a copy of the database management system. Failure of individual nodes in the cluster can be tolerated and the data can remain accessible. Such high availability and fault-tolerance is a primary reason for deploying a parallel database system, in addition to improved performance.

2.3.3 High Availability through Virtualization

Exploiting virtualization to provide HA enables us to decrease the complexity of the system because virtual machines are easier to configure and manage. At the same time, using virtualization for HA lowers the cost of HA because less hardware is required. A key technology that comes with virtualization is the ability to migrate virtual machines from one physical host to another while they are running. This capability is known as live migration [13]. Examples of commercial and research prototypes that use virtualization to provide high availability for applications running inside a VM include the following:

1. **VMware HA:** VMware HA is a commercial offering that is part of VMware Infrastructure 3 and provides automatic failover and restart for applications running inside a virtual machine.
2. **Remus [14]:** Remus is a research prototype that provides whole virtual machine replication with the open source Xen in an application and operating system agnostic manner, running over commodity hardware. No external state is ever lost, and the virtual machine is not restarted after a failure.Instead, it continues execution from where it left off before failover.
3. **everRun VM:** This technology provides HA for Xen Server virtualization. It provides automated fault detection and failover for individual VMs. Furthermore, it offers various levels of availability that can be selected on a per VM basis.
4. **Microsoft Hyper-V [15]:** Hyper-V by Microsoft can use Window Server 2008 failover clustering to provide HA for VMs.

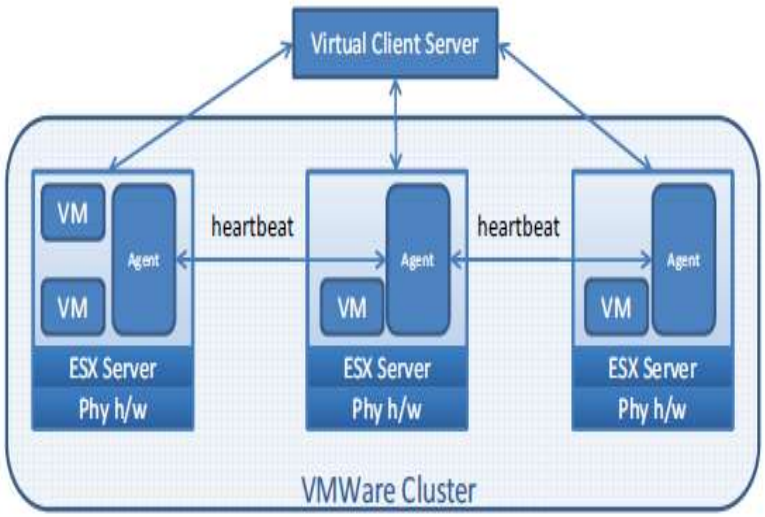

Fig. 2.3 VMWare System Architecture

2.4 Heartbeat for high availability

The first versions of heartbeat supported monitoring between two servers only, where one of them (the active one) was used regularly for serving. The passive one would serve as a backup server and all the time check the status on the active server by monitoring the heartbeats. If the active server stopped sending heartbeats, the passive backup server would take over the IP address of the active one and continue to serve the same content. The passive server now became the active one. If the failed active server came back online, heartbeat would negotiate a failback of the service, so that the initial passive server let back the control to the active one. [16]

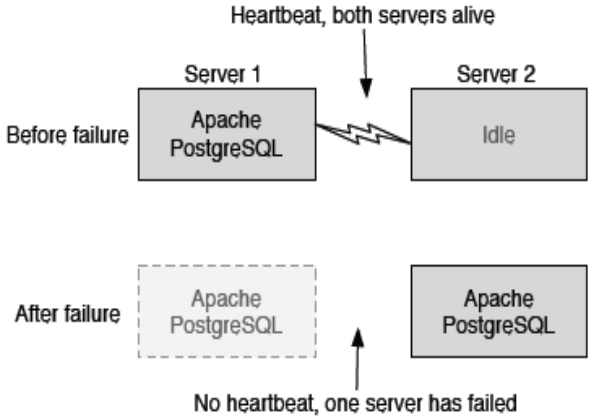

Fig. 2.4: Failover in an active/passive configuration with two servers. Only one server serves at all times.

A more advanced usage is having two active servers, each serving different content or services. A very typical example here is one server (a) that is running an SQL server, and one server (b) that is running an apache web server. They are both active since they both are serving. In case of failure of one of them, i.e. if server (a) dies, server (b) would start mysql in addition to apache so that it serves both services at the same time. The necessary changes to the configuration happen automatically. If server (a) resurrects, server (b) would stop its mysql server and server (a) would start it once again. This feature is called autofailback [16]

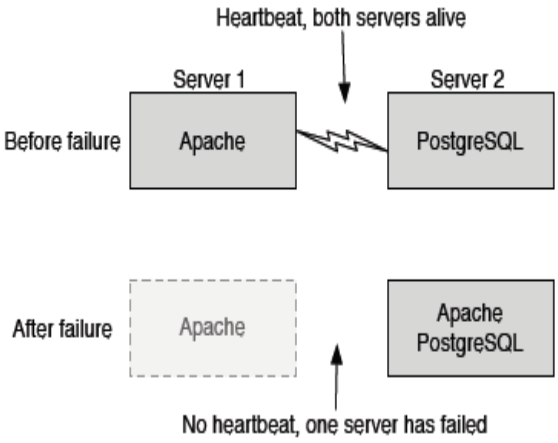

Fig. 2.5: Failover in an active/active configuration with two servers. Both servers serve initially.

Case Study (Amazon)

Amazon RDS provides high availability and failover support for DB instances using Multi Availability Zones deployments. Amazon RDS supports DB instances running several versions of PostgreSQL. You can create DB instances and DB snapshots, point-in-time restores and backups.

The primary DB instance is synchronously replicated across Availability Zones to a standby replica to provide data redundancy, eliminate I/O freezes, and minimize latency spikes during system backups.

In the event of a planned or unplanned outage of your DB instance, Amazon RDS automatically switches to a standby replica in another Availability Zone if you have enabled Multi-AZ. The time it takes for the failover to complete depends on the database activity and other conditions at the time the primary DB

instance became unavailable. Failover times are typically 60-120 seconds. However, large transactions or a lengthy recovery process can increase failover time. When the failover is complete, it can take additional time for the RDS console UI to reflect the new Availability Zone.[17]

III. CONCLUSION

There is critical need for highly available architectures in today's enterprise information systems. Numerous HA product options and solutions are available for the Postgres Plus family of databases.

The Amazon cloud provides a unique platform for any RDBMS, including PostgreSQL. With capabilities that can meet dynamic needs, cost based on usage, and easy integration with other AWS products such as Amazon CloudWatch, the Amazon cloud enables you to run a variety of applications without having to manage the hardware yourself.

LITERATURE REFERENCES

[1]. Hewlett Packard. Virtualisation - it supply meets business demand. 5983-0462EEE. Rev. 1,
[2]. September 2005.
[3]. The PostgreSQL Global Development Group. PostgreSQL 9.3.5 Documentation - Appendix
[4]. D. SQL Conformance. http://www.postgresql.org/docs/9.3/static/features.html.
[5]. M. Sloman. Policy driven management for distributed systems. Journal of Network and
[6]. Systems Management, **2**:333, 1994.
[7]. D. Verma et al. Policy based sla management in enterprise networks. In Policy Workshop
[8]. 2001. Springer Verlag, 2001.
[9]. Peter S.Weygant. Clusters for High Availability - A Primer of HP Solutions. 0- 13-089355-2.
[10]. Hewlett-Packard Professional Books, second edition edition, 2001.
[11]. Mark Burgess. Analytical Network and System Administration. Managing Human-Computer
[12]. Networks. 0-470-86100-2. JohnWiley & Sons, Ltd, 2004.
[13]. Jim Gray and Daniel P. Siewiorek. High-availability computer systems. IEEE Computer,
[14]. 24(9):39–48, 1991.
[15]. Research Paper Brasted_Epsen
[16]. Hui-I Hsiao and David J. DeWitt. A performance study of three high available data
[17]. replication strategies. Distributed and Parallel Databases (DAPD), 1(1), 1993.
[18]. Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a
[19]. solution. In International Conference on Management of Data (SIGMOD), 1996.
[20]. Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. Concurrency Control and
[21]. Recovery in Database Systems. Addison-Wesley, 1987.
[22]. Paolo Bruni, Roy Cornford, Rafael Garcia, Sabine Kaschta, and Ravi Kumar. DB2 9 for
[23]. z/OS Technical Overview. IBM Redbooks, 2007.
[24]. Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian
[25]. Limpach, Ian Pratt, and AndrewWarfield. Live migration of virtual machines. In
[26]. Symposium on Networked Systems Design and Implementation (NSDI), 2005.
[27]. Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and
[28]. Andrew Warfield. Remus: High availability via asynchronous virtual machine replication.
[29]. In Symposium on Networked Systems Design and Implementation (NSDI), 2008.
[30]. https://www.microsoft.com/en-in/server-cloud/solutions/virtualization.aspx
[31]. UKUUG LISA/Winter Conference, High-Availability and Reliability. The Evolution of the
[32]. Linux-HA Project, Bournemouth, February 25-26 2004.
[33]. http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_PostgreSQL.html