WIRTSCHAFTSUNIVERSITÄT WIEN
MASTERARBEIT

Titel der Masterarbeit:
"Database Clustering mit Fokus auf PostgreSQL"

Englischer Titel der Masterarbeit:
"Database Clustering with focus on PostgreSQL"

VerfasserIn:	Dmitry V. Petrovsky, BSc. (WU)
Matrikel-Nr.:	0052912
Studienrichtung:	Wirtschaftsinformatik (Mag, WINF-M03)
Textsprache:	English
Beurteiler:	Univ. Prof. Dipl.-Ing. Mag. Dr. Wolfgang Panny
BetreuerIn:	Dipl.-Ing. Mag. Dr. Albert Weichselbraun

Ich versichere:
dass ich die Masteratsarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe. dass ich

die Ausarbeitung zu dem obigen Thema bisher weder im In- noch im Ausland (einer Beurteiler-in/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe. dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

Datum _____ Unterschrift _____

Abstract

This paper gives a complete overview of the existing clustering approaches, from theory to practice. It starts with an overview of general clustering technologies, provides high-level cluster terms definitions and outlines possible issues a database cluster may encounter. It then goes further and examines popular commercial database products (Oracle and DB2) as well as their Open Source counterparts (MySQL and PostgreSQL), outlining their features, replication methods used and pays special attention to concurrency control and locking. The thesis provides an in-depth analysis of both - commercial and Open Source solutions, highlighting their advantages and disadvantages in the light of implementing a cluster.

Apart from information safety and asymmetry the thesis also addresses a number of problems which could be solved by a database clustering and caching system, such as load balancing, transparent failover and defect nodes replaceability.

The primary focus of this work is on the research of the available clustering approaches and technologies in PostgreSQL. Therefore, the paper examines a number of commercial and Open Source frameworks, based on the pre-defined requirements. The contribution lies in a direct comparison between PostgreSQL clustering products, available features, in spelling out advantages and drawbacks of each solution and in making a product recommendation based on the possible area of usage. Based on the analysis, selected PostgreSQL clustering frameworks are installed, configured and evaluated. Using the theoretical material on database replication and the known problem described in this work, a number of tests were designed in order to evaluate and compare the frameworks in question. The results of the research conclude this work.

Kurzfassung

Diese Arbeit gibt eine komplette Übersicht über existierende Clustering-Technologien, von Theorie bis Praxisanwendung. Als Einführung wird ein allgemeiner Überblick über Clustering-Technologien gegeben. Danach werden high-level clusterrelevante Begriffe vorgestellt und mögliche in einer Clustering-Umgebung auftretende Probleme skizziert. Darauf folgend werden die populärsten kommerziellen (Oracle und DB2) als auch Open Source Produkte (MySQL und PostgreSQL) behandelt. Es werden Features und Replikationsansätze ausführlich beschrieben. Der Concurrency-Kontrolle und dem Locking wird dabei besondere Aufmerksamkeit gewidmet. Die Arbeit liefert eine detaillierte Analyse von beiden - kommerziellen und Open Source Datenbanklösungen und beschreibt ihre Vorteile und Nachteile im Bezug auf Clustering-Umgebungen.

Abgesehen von Informationssicherheit und Informationsasymmetrien wird auch eine Zahl von diversen Problemen behandelt, die durch den Einsatz vom Cluster- und Cachingssystem gelöst werden können: load balancing, transparent failover und die Austauschbarkeit von ausgefallenen Nodes.

Das primäre Ziel der Untersuchung liegt in Analyse von vorhandenen PostgreSQL basierten Clustering-Lösungen. Demzufolge wird eine Reihe von kommerziellen und Open Source Frameworks in Betracht gezogen. Der Arbeitsbeitrag liegt in einem direkten Vergleich zwischen diversen PostgreSQL Clustering-Produkten und ihren Features. Die Vorteile bzw. Nachteile der einzelnen Clustering-Lösungen werden genau erläutert. Basierend auf dieser Analyse wird eine Produktempfehlung abhängig von möglichen Einsatzmöglichkeiten abgegeben. Basierend auf dieser Analyse werden ausgewählte PostgreSQL Clustering Frameworks installiert, konfiguriert und evaluiert. Anhand von der in der Arbeit vorgestellten Theorie über die Datenbankreplikation und damit verbundenen Problemen, wurde eine Zahl von Tests entwickelt, um die genannten Frameworks besser evaluieren und vergleichen zu können. Die Ergebnisse der Recherche schliessen diese Arbeit ab.

Contents

Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
ANSI	American National Standards Institute
API	Application Programming Interface
B2B	Business-to-Business
BSD	Berkeley Software Distribution
C2B	Customer-to-Business
CGE	Carrier Grade Edition
CS	Cursor Stability
DBMS	Database Management System
DDL	Data Definition Language
DML	Data Manipulation Language
DSS	Decision Support System
GIS	Geographic Information System
GiST	Generalized Search Tree
HADR	High Availability Disaster Recovery
IDIOM	Information Diffusion across Interactive Online Media
ISO	International Organization for Standardization
IT	Information Technology
JDBC	Java Database Connectivity
LCR	Logical Change Record
LDAP	Lightweight Directory Access Protocol
LOB	Large Object
MD5	Message-Digest Algorithm 5
MQ	Message Queue
MS	Microsoft
MVCC	Multi-version Concurrency Control
NDB	MySQL In-memory Clustered Storage Engine

OLTP	Online Transaction Processing
OS	Open Source
OSS	Open Source Software
PAM	Pluggable Authentication Modules
PgQ	Generic High-Performance Queue for PostgreSQL
PgSQL	Loadable procedural language for the PostgreSQL
RAC	Real Application Cluster
RBR	Row Based Replication
RR	Repeatable Read
RS	Read Stability
SBR	Statement Based Replication
SE	Standard Edition
SQL	Structured Query Language
TCO	Total Cost of Ownership
TCP/IP	Transmission Control Program / Internet Protocol
WAL	Write Ahead Log
XML	Extensible Markup Language

List of Figures

List of Tables

Part I.
Theory

1. Introduction

A database management system (DBMS) is a set of software tools designed to manage a large unit of information which involves setting structures for data storage and also provides mechanisms for data manipulation. A DBMS must guarantee the safety and availability of the stored data, despite crashes or unauthorized access attempts (Tes07).

The IDIOM Media Watch on Climate Change relies on DBMS solutions as well. It visualizes contextualized information spaces comprising millions of documents, allowing a user to navigate through its repository alongside multiple dimensions and to formulate queries based on textual, semantic, or geospatial criteria. High performance database solutions are required to support real time browsing and searching of this vast document collection and are very important.

1.1. Database clustering and its importance

Business processes are enacted manually, guided by the knowledge of the company's employees. Additional benefits can be achieved if organizations use DBMS for coordinating the activities involved in their business processes (Wes07). The DBMS can separate data from applications which enables the design and development of business applications with business process separation in mind and makes such applications more flexible and less dependent on data changes. Trying to achieve the efficiency of business processes automation and this way to gain a competitive advantage more and more enterprises invest heavily in DBMS (GGK+05; Wes07). With the rapid development of B2B and C2B solutions the access to data plays a key role, therefore access to information within an organization is often vital for its business processes. The loss of the information (e.g., as a result of a natural disaster, a hardware failure or a simple mistake of an employee, etc.) may have unpredictable consequences (Tes07).

Database replication technology is the key to shared-nothing database clusters which are a popular solution to multi-tier applications (JP03). First approaches to database replication had a number of serious limitations, but the recent development in communication protocols and replication extensions made the way for new replication algorithms that are able to offer highly efficient, consistent, fully replicated and error prone systems (JP03). Such advances are now making the transition from the research labs to concrete products and applications and are the research subject of this research paper.

1.2. Paper outline

The thesis follows an incremental approach. The first chapter introduces high level elements of database clustering and shows how fully fledged database solutions could be built. It starts with the description of the main goals and challenges and provides a comprehensive overview of the main clustering approaches. Then it goes further and provides an overview of the existing database technologies and their clustering implementations, considering the potential benefits outlined above with a main focus on PostgreSQL.

The third chapter aims at providing a thorough overview of the available clustering approaches implemented in databases. It reviews the replication methods used, examines locking and concurrency support, outlines advantages and weak points. The most popular and widely used commercial and OS products are carefully examined and compared based on its features.

The fourth chapter introduces the available clustering frameworks based on PostgreSQL. Commercial and Open Source solutions are analyzed and reviewed. It outlines a number of criteria, how a suitable clustering solution should look like, defines the key-technologies and based on the pre-defined schema, the best solutions are chosen for testing and benchmarking.

The fifth chapter is concerned with an analysis of case studies. A simple cluster is designed, the selected PostgreSQL clustering frameworks are setup and tested. The tests benchmark results are introduced in this chapter as well. Finally, the author draws conclusions and gives an outlook to possible future developments based on the research made.

2. Clustering approaches

This chapter gives an introduction and outlines a definition to database clustering and replication. It provides an overview of the advantages, disadvantages and known problems of traditional solutions. It elaborates on the main drawbacks of the traditional approaches, underlining avoidable limitations, analyzes current trends and tries to find a state-of-the-art solution based on the research made in the area of the database replication.

There are lots of definitions what clustering is and a dictionary might describe clustering as a "group of related items" [1], what would be perfectly correct, but for a database we may rephrase it as "grouping of related items stored together for efficiency of access, resource utilization and information safety" (Ver).

The primary cluster components are processor nodes, a cluster interconnect and a disk subsystem. The clusters share disk access and resources that manage the data, but each distinct node does not share memory. While clustering is a very popular term, it does not have a well defined meaning with regard to database systems. Lots of different techniques, like replication, load balancing, distributed querying, etc., are called clustering here and there [2]. In general, a cluster is a real-time distributed transactional database designed for fast, always-on access to data under high throughput conditions. It consists of a number of interconnected by a dedicated high-speed network database servers (KGK95).

Each database server is called a node, meaning a database participating in replication, and depending on its function it could be further subdivided into the following categories (EN03):

- *a master node* is the main server which coordinates and monitors all other nodes and actions. Depending on an action triggered a master node fires off a responsible slave node to perform an operation. A

[1] http://www.merriam-webster.com/dictionary/cluster
[2] http://www.postgres-r.org/documentation/terms_3

master node may also be responsible for load balancing and query partitioning. In order to boost cluster's performance, it is advised to have a couple of master nodes.

- *a slave node* has a single function it is responsible for, it could be further subdivided into different categories, e.g.: Log Node, User Node, Data Node, etc. All slave nodes are coordinated by a master node. In most cases, a slave node is a data node, i.e., it is used to save the data. If a load balancer is configured, some slave nodes are used to serve read-only requests, while the other ones process write queries and then replicate the changes with the other nodes in the cluster.
- *a stand-by node* is a reserve server which does nothing. In case of a master node failure it starts off and takes the place of the master node.

A database cluster may have different constellations. Depending on needs or complexity of a particular project, a cluster may have different configurations (EN03):

- *master to multiple slaves* is the simplest and most common configuration. A master server is used to manage its slave nodes.
- *multi-master to multiple slaves* – if high-availability is an issue, then multi-master architecture is the right way to go. In case a master-server goes down, another one takes its place, thus ensuring the further cluster availability.

2.1. Introduction to Database Replication

There are two known kinds of database replication: full replication, when each node has a full copy of the database and partial replication, when parts of a single database are distributed between the nodes (Mat97). The access to the database is implemented using transactions which are read or write operations. The essential part of a cluster implementation is concurrency and replica controls. Concurrency control provides each transaction with

the needed isolation level and replica control manages the access to the nodes in a cluster.

Another important point to consider while implementing a clustering solution is the data consistency. In this respect, there are some criteria which specify the data's correctness. The strongest one is 1-copy-serializability (BHG87), i.e., although each node stores separately a copy of a database, each copy is identical to each other and access to any node in a cluster returns the same result. Transaction atomicity has to guarantee that the commit operation applies changes to all nodes or none. It has to be noted though, that due to performance considerations not all protocols can guarantee atomicity.

The changes could be applied to the nodes in a cluster within a transaction, in this case the replication is eager, or after it commits, meaning the replication is lazy (BK97). Each of these approaches has its advantages and disadvantages. The first one insures a higher degree of data consistency and can detect possible conflicts before the commit operation, but results in a significant and very expensive communication overhead within the transaction. The lazy replication on the other hand consumes changes after the end of the transaction which might cause some data inconsistencies (BK97).

There are two known approaches to the nodes replication - primary copy, when all updates are executed on primary server first, and a distributed approach which allows any copy to be updated. The first approaches simplifies the concurrency control substantially, but may also result in a single point of failure. The lazy replication may cause a number of problems: when a couple of transactions update different copies of the same data and commit locally, it results in a data inconsistency (BK97).

2.2. Eager vs. Lazy Replication

To implement eager replication is quite straightforward: for example a 2-phase-locking, timestamp based algorithms and 2-phase-commit could be

used to guarantee serializability and atomicity. The majority of algorithms implementations try to avoid centralized primary copy approach in favor of the update everywhere approach, e.g., *read-once/write-all* requires updates to access all the nodes while read operations are executed locally. Such an approach is considered to be not very reliable, as an update process freezes if one of the nodes is not available. Therefore another approach is more favorable in this respect - *read-one/write-all-available* which executes updates only on the available nodes (BHG87).

There are a number of quorum protocols, requiring both operations (read and write) to access a quorum of nodes (AEAS97; HSAA99). As soon as a quorum of available nodes agrees on execution, the operation can be processed. On the other hand, using an optimistic approach suggests to execute transactions locally and broadcast the changes in some ordered form. In case of a conflict, a causing transaction has to be rolled back. Such an approach is called an epidemic update propagation and is very similar to a 2-phase-commit. Furthermore, multicast primitives with different ordering could be used for updates propagation which results in a reduced number of roll back operations (SAA98).

The eager replication approach guarantees the data consistency, but despite of this fact, it is not widely used. Oracle Advanced Replication implements an eager protocol, using stored procedures and triggers. As soon as an update is applied locally, a stored procedures fires up an event and launches a synchronous replication which locks the corresponding remote entries. Oracle itself recommends to use eager replication only if the data consistency has the highest priority and therefore it should be avoided in the majority of cases as it is highly dependent on network availability (Tum04).

Lazy replication dramatically reduces the transaction execution time, as updates are committed locally and only sent to the other nodes. Some lazy replication implementations can only ensure that all nodes eventually converge to a single final value, regardless of the transaction dependencies. Such an approach could not guarantee atomicity in any case, i.e., in case a node fails before the updates of a committed transaction are propagated, the

transaction gets lost. The majority of the lazy replication implementations use a primary copy approach - updates must be executed on a master node and then propagated to the slaves, the direct slave nodes updates are not allowed at all. The huge disadvantage of this approach which could not be avoided though, is that all the nodes could not be kept synchronized at all times.

Using lazy replication, serializability cannot be guaranteed in every case, but the recent researches in this area provide a possible solution to this issue (CRR96). The main idea is the primary and secondary copies allocation, using configuration graphs where there is a non-directed edge between two nodes if one has the primary copy and the other a secondary copy. If it is an acyclic graph, then the serializability can be achieved by updates propagation on transaction commit (CRR96). This approach was enhanced by allowing certain cyclic configurations which however require more complex update propagation algorithms, e.g., updates to secondary copies must be applied causal (PSM98; PMS99). An alternative solution is to eliminate cycles - edges are directed from a master to a slave which leads to the introduction of sophisticated update propagation strategies, e.g., to transform graphs into a tree thus a master is not required to be directly connected with all its slaves. The updated are applied along the paths of the graph (BKR+99). On the other hand, each transaction could be assigned a timestamp, this way a total order of to be executed transaction is defined. Another proposed strategy is to combine lazy with eager propagation and to use lazy propagation along the acyclic paths and eager replication for cyclic paths (BK97; ABKW98).

2.3. Replication in a cluster

The goal of using a database cluster is to distribute the workload among the nodes and to achieve scalability and fault tolerance. As the workload increases, new nodes are added to adapt the system performance accordingly. In case one node goes down, it does not effect the system availability

as a whole, the other nodes take over the work over the failed ones. Besides work load distribution, a database cluster could be used to partition the data among the sites, where each site is responsible for execution of queries on specific data. The known limitation in this respect are: in the first place, it is not easy to divide the data and to distribute the work load accordingly. Another problem to consider is transaction management, e.g., in case a transaction wants to access data on different partitions. The more serious situation arises, when individual nodes responsible for a particular data partition fail which results the data inaccessibility. In order to avoid some of the above listed problem, data replication is required. The following strategies could be followed:

- *load balancing* – the sole purpose of a load balancing, as the name suggests, is to adjust and coordinate the load between database instances. On the other hand, it could be seen as a security layer as well, e.g., logically and physically dividing database instances to the ones which store only read-only data and the ones with read-write access (FAA99).
- *fault tolerance* – using eager replication a system availability could be guaranteed, as long as at least one node is accessible.
- *data consistency* – database clusters have the goal to manage high volumes of data, therefore a proper handling of inconsistencies is extremely important and could not be done manually. The replica control has to detect and eliminate any inconsistency automatically and transparently.
- *transaction management* – all the updates have to be executed on all dependent nodes. If this is not the case, individual nodes must be able to subscribe, receive and apply the necessary changes fast at any time.

Considering these strategies, it is obvious that the eager replication is the most desirable one in the light of cluster computing, it guarantees data consistency, enables fault tolerance and at the same time simplifies workload distribution, however it suffers a severe performance hit (BKR+99).

2.4. Known problems of traditional approaches

Eager replication sounds like the first choice when it comes to database replication, at least in theory. The major drawback is the huge performance penalty. The interesting point here is the limitations of eager replication and whether it is possible to avoid them. We are going to analyze and discuss traditional replication approaches, their impact on cluster performance and complexity, and the possibilities of avoiding the known issues by use of an adequate technology.

2.4.1. Conflicts Management

One of the main problems of eager replication is the conflict rate and a possibility of deadlocks. The approximate estimates of deadlocks in different scenarios is directly proportional to n^3, where n stands for number of nodes (GHOS96). The more transactions accessing the same data and the more nodes exist in the system, the more time is required to lock the corresponding entries, resulting in a longer transaction execution time. Another point to mention is that transactions execution time increases due to communication overhead which by itself may cause conflicts and deadlocks.

Transactions pre-locking techniques could be used to reduce the rate of conflicts and possibly to avoid deadlocks, e.g., to use group communication systems (HT93): the idea is to multicast the messages which ensures that the same total number of messages is distributed within the communication group. Therefore, the messages are sent in a single operation and are received by all the nodes in the same order. Locks are granted in the order the messages arrive which guarantees the same updates on all nodes in exact order. If a node goes down, it would still receive the update messages in the right order, as soon as it becomes available. More than that, in this way the deadlock problem could be solved (HT93; BKR+99).

2.4.2. Communication Overhead

The most protocols execute the incoming updates individually, i.e., each message triggers two messages - a request and its acknowledgement, what results in 2^n number of messages. The problem of this approach is quite obvious, the more nodes are in a cluster, the more messages are being sent. For instance, if a cluster consists of 10 nodes and an average number of transactions which take place per second is 100, and each transaction has five update operations, this would lead to 9.000 messages per second in a point-to-point connection. Starting a transaction a master node has to wait until all the acknowledgements from all nodes participating in a transaction are received, i.e., the transaction is stalled. Thus such an approach would produce a huge amount of traffic, increase complexity and dramatically damage the performance of a cluster.

To minimize the overhead the write operations eager replication algorithms bundle write operations in a single write set message instead (very much like lazy replication approach) and move it to the end of the transaction. Such a technique greatly improves the message throughput and increases the performance in general.

2.4.3. Transactions and Isolation levels

The levels of transaction isolations are specified in the SQL standard (KKH08; BBG+95). There are a number of isolation levels databases are using. Generally, it could be seen as a trade-off between performance and data correctness in order to reduce the possible conflicts between transactions. In some cases significant performance boosts could be achieved at the price of temporary data inconsistencies. The supported isolation levels varies from database to database, therefore they are separately discussed in the next chapter.

2.4.4. Fault tolerance

Fault tolerance is one of the critical issues for database clustering and it introduces a considerable amount of overhead in order to provide it. The majority of traditional approaches use commit protocols to guarantee the atomicity of a transaction. Such a commit protocol has a big impact on the cluster performance, as the transaction execution time is equal to the response time of the slowest node. The nodes in a cluster agree to commit a transaction when all of them can guarantee a local commit which is the point when a transaction is being executed. Because of this, eager replication is used with a dedicated hardware equipment in dedicated network environments, where network availability could be guaranteed.

In order to avoid such a problem and at the same time to improve cluster performance, it is recommended to weaken full correctness checks. Thus a local commit is allowed independent if other nodes could execute a transaction or not. This way, the local transaction is not required to wait for other nodes to acknowledge the request and commit a transaction. The master node optimistically assumes that other nodes in a cluster will serialize the transaction the same way (AAA+96).

2.5. Summary

The major terms, definitions, techniques and possible areas of problems used in a cluster environment were introduced in this chapter. We gave a definition to a cluster, described underlying technologies and underlined the differences in between traditional approaches. Very generally we have also outlined the possible areas of problems connected with database replication based on a replication method used. It has to be noted though, that it is rather not possible to provide a common solution to the described problems, but rather the whole cluster configuration and the underlying database has to be taken into consideration, as the differences in implementations are quite huge. Therefore, a detailed database solutions overview is required

in the first place which is given in the next chapter. The problems described in this chapter are addressed in the chapter "Evaluation", where different frameworks using both lazy and eager replications are tested and compared.

3. Implementations

There are a huge number of database solutions on the market today and with the recent advances in the OS world it is not necessary to invest a fortune to build a database cluster. But it is rather possible to take advantage of OS solutions, download a product, set-up a custom cluster and test it thoroughly before using it in production environment. The online documentation, community support and product source code make it all possible to use OS products not only for private purposes but for a commercial implementation as well.

Still the commercial products from Oracle[3] and IBM[4] are considered to be market leaders in this area. They claim to provide the best-practice clustering products available on the market today, listing the rich set of supported features which would meet practically all possible requirements. The goal of this chapter is to research, investigate and compare the commercial and OS solutions, to outline the areas of usage, advantages and drawbacks in the light of an enterprise, i.e., cluster environment.

First, we are going to introduce the most popular commercial products followed by their OS counterparts. In order to make a plausible and fair review, we are going to define an analysis schema for the research:

- a brief description introduces a database product, gives a short description and depicts the manufacture slogan.
- a features overview provides the general feature overview with the main focus on the clustering technologies.
- replication technical details provides technical details on the replication technologies used in the framework in question. Describes the limitations and underlines advantages of the framework.

[3] http://www.oracle.com/database/index.html
[4] http://www-01.ibm.com/software/data/db2/

- locking & transaction management introduces the locking and concurrency implementation.

A short product description comes first, it outlines the database target audience, its purpose and some particularities. The features overview discusses supported clustering features and technologies, the differences and similarities to the other reviewed products are stressed.

The chapter starts with the analysis of the most advanced enterprise products in the commercial world: DB2[5] and Oracle[6], giving an overview based on the schema defined above. Then it introduces the most commonly used OS solutions, such as MySQL Cluster[7] and PostgreSQL[8], highlighting their approaches, advances and philosophy, outlining the major features and comparing the frameworks not only to the commercial products but also to each other.

Finally, a feature-based table compares each examined product and summarizes the analyzed products' suitability or non-suitability for building a cluster for a concrete production environment. Therefore, it is essential to define, what the production environment is. Thus the cluster has the following characteristics:

- a high-performance and scalable cluster – it should be able to manage huge loads of data.
- 24/7 high availability – it should be possible to provide non-stop data availability.
- data consistency and integrity at any time – information consistency should be guaranteed.

[5] http://www-01.ibm.com/software/data/db2/9/edition-enterprise.html
[6] http://www.oracle.com/database/enterprise_edition.html
[7] http://mysql.com/products/database/cluster/
[8] http://www.postgresql.org/

3.1. Commercial products

Oracle and IBM both provide a number of database solutions, not only for commercial use but also free ones. The free of charge products are the entry-level database offerings, i.e., a developer may download it, use it, deploy and distribute his application free of charge. The source code, as one would expect, is not provided, the products are distributed in binary format and are pre-compile for a number of platforms. Such products are generally used for demonstration purposes, to familiarize a developer with the product line and have a number of limitations, e.g., as in case with Oracle, it can only use up to 4GB of user data, use up to 1GB of memory, and use only one CPU on the host machine[9]. These are serious limitations, so the entry level solutions could not be used in any serious project. The clustering technology is not supported at all.

In order to overcome such limitations and to acquire additional features (e.g., to build a cluster), a developer is required to download a full version of the software and acquire a license. Depending on the required features one may wish to use, the number of concurrent connections and the size of a database, the license costs may add up quite quickly (Oraa). It must be also noted, that the license costs are to be payed annually. The decision to adopt a database solution has to be analyzed very careful as it is very challenging, time consuming or sometimes even impossible to switch a database (EN03).

Under commercial products discussed in this chapter, we mean the enterprise products from Oracle and IBM - Oracle RAC and DB2 Enterprise Server Edition respectively. These products have all the defined characteristics of a production clustering database.

[9] http://www.oracle.com/technology/products/database/xe/index.html

3.1.1. Oracle Real Application Cluster

Oracle describes its product as: "*...an option to the award-winning Oracle Database Enterprise Edition. Oracle RAC is a cluster database with a shared architecture that overcomes the limitation of traditional shared-nothing and shared-disk approaches to provide a highly scalable and available database solution for all your business applications. Oracle RAC provides the foundation for enterprise grid computing.*".[10] Oracle supports a wide range of platforms, provides a very detailed documentation and excellent support (Oraa). The database product solutions from Oracle have a long history, are considered to be very reliable and are widely used in the production environment. The installation is quite straight forward with the step-by-step instructions, although the list of pre-requirements, hardware- and software-wise is quite challenging. It is SQL-standard compliant, shines with its advanced command line interpreter and GUI interface. It is a highly customizable and feature-full product which supports not only standard features, but goes far beyond them. Oracle database is viewed as an industry standard for a good reason.

Features overview

Oracle RAC supports all the standard clustering features described in the previous chapter (shared nothing, shared disk approaches, eager and lazy synchronization, etc.) by default plus additionally (Oraa):

- Oracle RAC provides single image installation and management
- the transparent deployment of a single database across a cluster of servers
- it also supports mainstream business applications of all kinds including OLTP, DSS and Oracle's unique ability to effectively support mixed OLTP/DSS environments (including popular packaged products such as SAP, PeoplSoft, etc).

[10] Oracle Data Sheet, http://www.oracle.com/technology/products/database/clustering/pdf/ds_rac11g.pdf

- the product comes bundled with a complete integrated Clusterware solution which includes mechanisms for cluster messaging, locking, failure detection, and recovery
- it provides Enterprise Grids, used for dynamic resources allocation
- 3rd party clusterware software is supported as well, although not needed
- it includes a High Availability API

Besides of the standard features and the ones listed above, Oracle RAC delivers a complex enterprise solution which provides an advanced set of tools for out-of-the-box transparent cluster implementation and replication. None of the reviewed systems delivers such an advanced clustering solution. It has though its price, and for a majority of enterprises the resulting costs are a way far from their budget (Oraa).

Replication technical details

To build a cluster Oracle RAC depends on Oracle Clusterware which coordinates multiple database server. Oracle Clusterware enables the database nodes to communicate with each other. It forms the cluster and manages the nodes as single logical server. Two key components are used to manage the cluster: Oracle Cluster Registry which records and maintains the cluster and node membership information and a voting disk which acts as a tiebreaker during communication failures. Consistent heartbeat information from all the nodes is sent to the voting disk when the cluster is running (Wik).

Oracle RAC addresses several areas of database management, such as:

- load balancing, connection pooling and query partitioning
- fault tolerance
- scalability

Two different replication forms are supported by Oracle (Orab):

- *basic replication* is implemented using CREATE SNAPSHOT or CREATE MATERIALIZED VIEW statements. It can replicate data only, no indexes or stored procedures could be replicated. The created snapshots are read only.
- *advanced replication* supports various configurations. It is more difficult to configure but allows to replicate not only data but also database objects such as indexes and procedures.

Oracle does not support replication to non-Oracle databases. It also does not support the replication of sequences, LONG and LONG RAW data types. In the latter case, LOB data type could be used instead.

The replication itself is being implemented using Oracle Streams which is the key technology for information sharing. It captures and distributes database updates, events and application messages which are automatically applied to destination nodes or passed to custom procedures and applications. The Oracle Streams architecture is shown in the following Figure 1

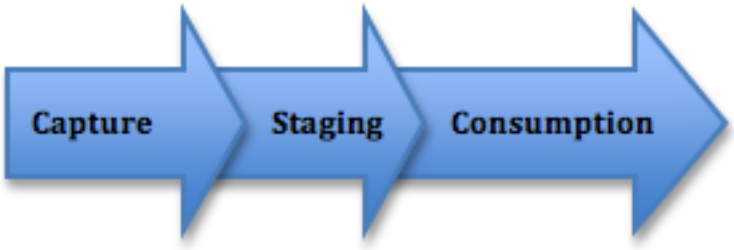

Figure 1: Oracle Streams Architecture (Orab)

Oracle Streams consists of three main elements: capture, staging and consumption. Figure 2 illustrates the replication flow. Oracle Streams monitors the changes made to a source database and replicates those to one or more remote nodes. Using Oracle Stream terminology, it captures changes, stages those changes and then consumes them at each database node (Oraa):

- **Capture** - the events are captured whether explicitly or implicitly, i.e., applications are allowed to generate events and place them in the

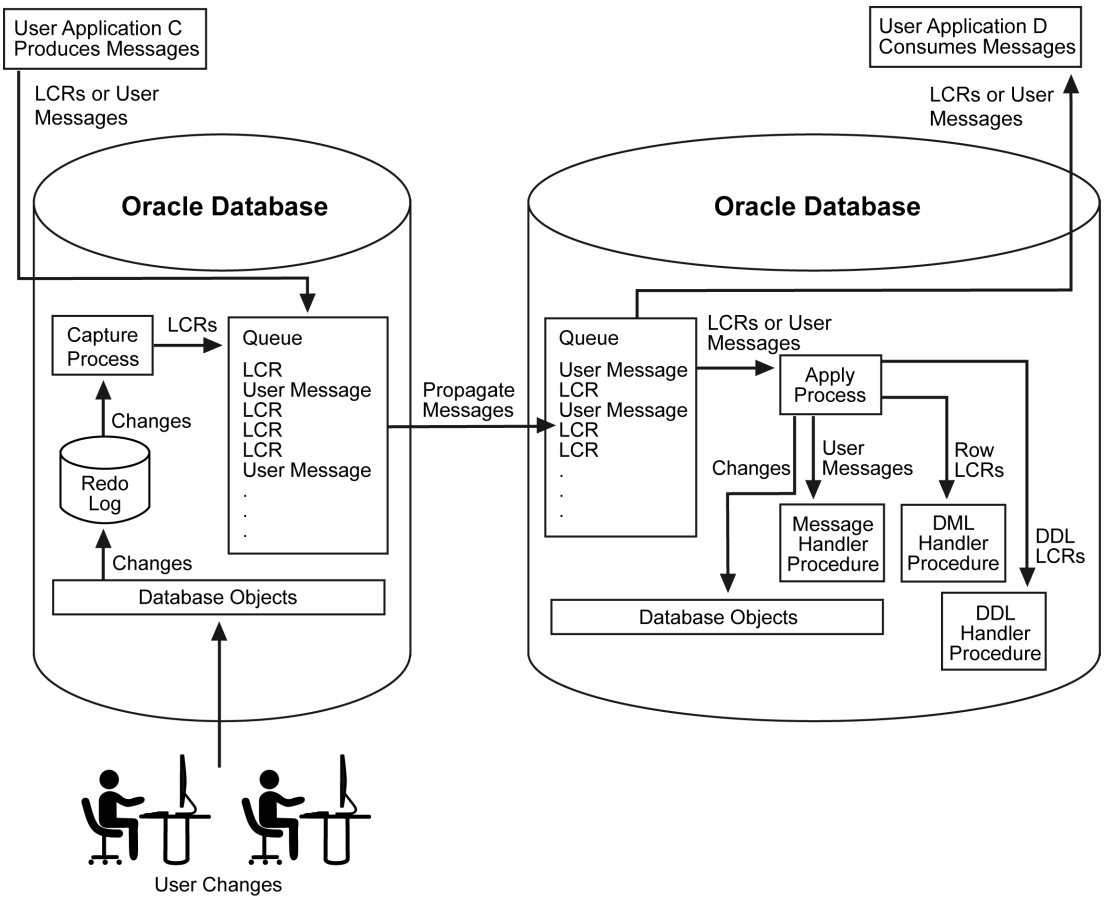

Figure 2: Oracle Streams Replication (Orab)

staging area or events are being captured automatically using one of the following techniques: log-based capture or synchronous capture. The server captures DML and DDL events by examining the redo logs, log buffers, and archive logs locally. Alternatively, the synchronous capture observes DML changes by monitoring transaction activity, i.e., the changes are captured as they happen. Independent of which capture method is used, the changes are converted to a Logical Change Record, i.e., a change to a single row. In case an SQL statement effects multiple rows, multiple LCRs are generated and put into a persistent staging area or queue (Orab).

- **Staging** - the database changes, already formatted as LCRs, are placed

in a staging area which is a persistent queue designed to store captured events. The queue provides a holding area with security, auditing and data tracking event features. Subscribers examine the contents of the queue and decide whether or not to take an action on the event. In the role of the subscriber could be an application, another queue or a default apply process. A subscriber may define and evaluate a set of rules to determine whether the events in the queue meet its criteria and if an action is to be triggered and the event in question to be consumed (Orab).

- **Consumption** - to process the events from the staging queue, Oracle Streams uses the apply engine. The changes are read from the queue and applied to a database or consumed by an application. The engine processes DML, DDL, user-supplied LCRs and user-enqueued messages. Apply engines support default and custom apply procedures. The first one automatically applies DML, DDL and user-supplied LCRs. The engine can also handle conflicts and invoke a resolution routine. *Customized apply* lets the database administrator control the apply process. Oracle Streams can register custom PL/SQL procedures and invoke those during the application process. A user may define multiple handlers if needed which are then called to manage the replication process (Orab).

Oracle Streams has the goal to simplify the data sharing between databases and database clusters, it unifies two approaches: message queuing and data replication capabilities. Oracle Streams replication provides rich, flexible and sophisticated configuration choices to handle the real world situations. The technology is designed to handle the data sharing in complex distributed database environments.

Locking & transaction management

Oracle's locking & transaction management is a golden standard in the industry. The ANSI/ISO standard defines four isolation levels (Kyt05). These levels are defined in terms of three phenomena that are either permitted or not at a given isolation level (Tum04):

- *dirty read* – reads of uncommitted data are allowed. Information integrity and consistency is not guaranteed.
- *non-repeatable read* – at different points in time, the content of a read row may change.
- *phantom read* – executing a query at different points in time, may deliver different results. The main difference to a non-repeatable read is, that the data already read can not be changed.

The SQL standard does not impose a specific locking scheme or mandate particular behaviors, but rather describes these isolation levels in terms of these phenomena, allowing for many different locking/concurrency mechanisms to exist (see Table 1)[11].

Isolation level	*Dirty Read*	*Non-repeatable Read*	*Phantom read*
Read uncommitted	Permitted	Permitted	Permitted
Read committed	–	Permitted	Permitted
Repeatable read	–	–	Permitted
Serializable	–	–	–

Table 1: Oracle isolation levels (Oraa)

Oracle explicitly supports the READ COMMITTED and SERIALIZABLE isolation levels as they are defined in the standard. However, the SQL standard delivers various degrees of consistency at each isolation level: REPEATABLE READ, according to SQL, guarantees a read-consistent result from a query. READ COMMITTED on contrary does not return consistent

[11] http://www.oracle.com/technology/oramag/oracle/05-nov/o65asktom.html

results, and READ UNCOMMITTED is the level to use to get nonblocking reads (Oraa).

Oracle Database has some particularities in this respect: READ COMMITTED has all of the attributes required to achieve read-consistent queries, while in some other databases, READ COMMITTED queries will return answers that never existed in the database. Oracle Database does not need to support dirty reads in order to provide non-blocking reads, as the other databases do. Oracle database provides nonblocking reads by default.

Additionally Oracle Database provides a READ ONLY transaction level which is equivalent to a REPEATABLE READ or SERIALIZABLE transaction and cannot perform any modifications in SQL. A transaction using a READ ONLY isolation level sees only those changes that were committed at the time the transaction began. Inserts, updates, and deletes are prohibited in this mode (Oraa).

Within a transaction a Save-Point could mark the point of safe rollback. Oracle RAC supports a distributed transaction (requires two or more nodes) and a two-phase commit which guarantees that all database servers participating in a distributed transaction either commit or rollback a transaction[12].

3.1.2. DB2 Enterprise Server Edition

DB2 is an alternative product developed by IBM to compete with Oracle. It is very similar to Oracle in the sense of provided features and documentation. It provides all the tools necessary for cluster deployment. The list of pre-requirements is a way much longer that the one of Oracle. IBM recommends using DB2 with AIX operating system using their own hardware which brings a number of advantages. It is no surprise, that IBM tunes its solutions for their own hardware platform, encouraging enterprises to buy the hardware as well. The only point of disappointment at this stage is

[12] Oracle Database Administrator's Guide

the command line environment used in DB2. In comparison to other reviewed products, it is very outdated. Although IBM does provide a number of GUI Tools for DB2 monitoring and controlling, it would be still a very welcomed feature to have a powerful command line interpreter for remote administration.

Features overview

DB2 enterprise server supports all the standard clustering features the same way Oracle does. It delivers a rich number of tools for cluster controlling and monitoring. Apart from it, the following key technologies are used in the product (IBMb):

- High Availability Disaster Recovery
- Tivoli System Automation
- table partitioning
- multi-dimensional data clustering
- full intra-query parallelism
- a set of performance optimization tools

Both (Oracle RAC and DB2 Enterprise Edition) provide advanced clustering solutions and all the necessary tools. One point which speaks for IBM in comparison to Oracle is the pricing model. Compared to Oracle RAC, DB2 Enterprise Edition is a bargain[13].

Replication technical details

DB2 uses the DB2 high availability instance configuration utility (db2haicu) to configure and manage database clusters. The utility has the following tasks:

- add and remove databases to or from cluster domain
- identify primary and standby nodes

[13]http://www-01.ibm.com/software/data/db2/9/edition-enterprise.html

- specify failover policies

The DB2 cluster manager, based on defined configuration and specified rules, manages the database instances in the clustered environment. The software is responsible for load balancing, connection pooling and failover management. The data replication between database instances in the cluster is managed by IBM WebSphere Replication Server which highly relies on WebSphere Message Queue for subsequent message processing. The message queue technology used by IBM for data replication is very similar to the approach used by Oracle.

DB2 provides two different solutions to the replication: SQL replication and Q replication. In SQL replication, committed changes are stored in relational tables before being sent to target systems. During Q replication, the changes are saved in messages that are then transported via WebSphere MQ queues to target systems (a similar technology is used by SkyTools which uses PgQ for message transporting).

SQL replication is used for a wide number of reasons, e.g., backups, capacity relief, and auditing change history. It is possible to set the replication interval to continuous, intervals, or for one time only. Continuous replication is used when applications need real-time data, i.e., clustering solution, such as airline reservations, bank transfers, etc. DB2 can replicate not only to another DB2 databank, but also to non-DB2 relational databases, such as Informix, MS SQL Server, Oracle and Sybase. Additionally, DB2 is very flexible to what kind of data is to be replicated - a user may define a subscription set(s), e.g., all rows and columns or just a subset of these. It is also possible to define rules determining how the data is replicated. These rules clean, aggregate or manipulate the distributed data to any extent. A user may also define which nodes get the manipulated data and which not. Figure 3 illustrates a simple SQL replication flow (IBMb).

On the other hand, Q replication is used for copying large volumes of data at low levels of latency. The changes are captured and converted to messages.

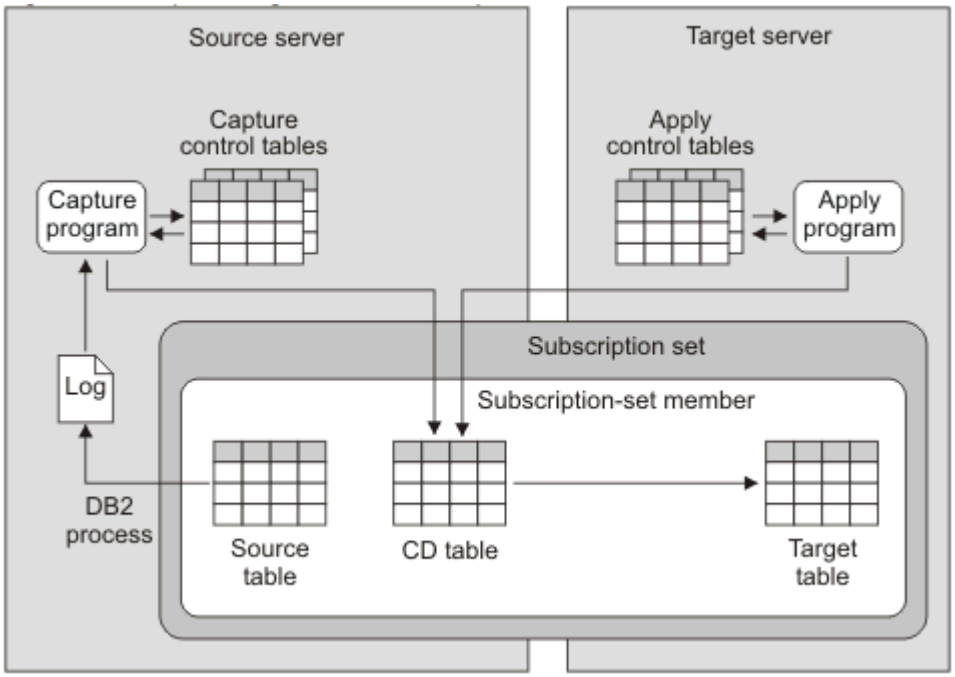

Figure 3: DB2 simple SQL replication (IBMb)

As soon as the data is committed and read by Q replication, it is sent to the target servers through WebSphere MQ. At the target nodes, the messages are read in the same order and converted back into transactional statements which are then applied to the target tables. Using MQ a user may be sure, that the data once read by Q, is going to be delivered and applied to the target table in the right order. Replication sets and rules could be applied here as well, in the same way it is used for SQL replication. Figure 4 shows a simple configuration in Q replication (IBMb).

DB2 provides one more replication method which is called event publishing - the changes to source tables are captured and converted into XML messages which are then put on WebSphere MQ message queues. The messages are read by a message broker or other applications and could be published anywhere a user sees fit. Such a method is a reasonable solution for feeding the data to information brokers and Web application. The event publishing

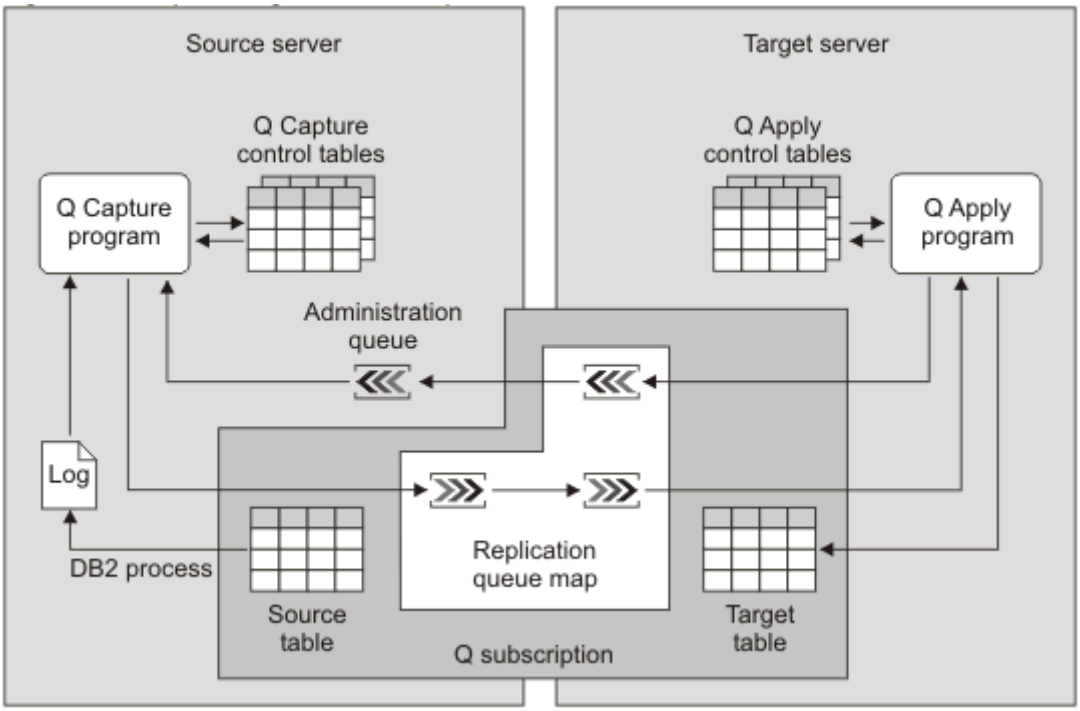

Figure 4: DB2 simple Q replication (IBMb)

replication is illustrated in Figure 5 (IBMb).

Table 2 provides a brief comparison between SQL and Q replications implemented in DB2 (IBMb):

Point of comparison	*SQL replication*	*Q replication*
Replication objects	tables and views	tables and stored procedures
Pairing of sources and targets	one or more replication sets	Q subscription map
Grouping	grouping source-target pairs into subscription sets	Q subscriptions are grouped by queue
Subsetting of columns and rows	Yes	Yes
Data transformation	Stored procedures, SQL statements	Stores procedures or programm calls

Table 2: DB2 SQL and Q replication comparison (IBMb)

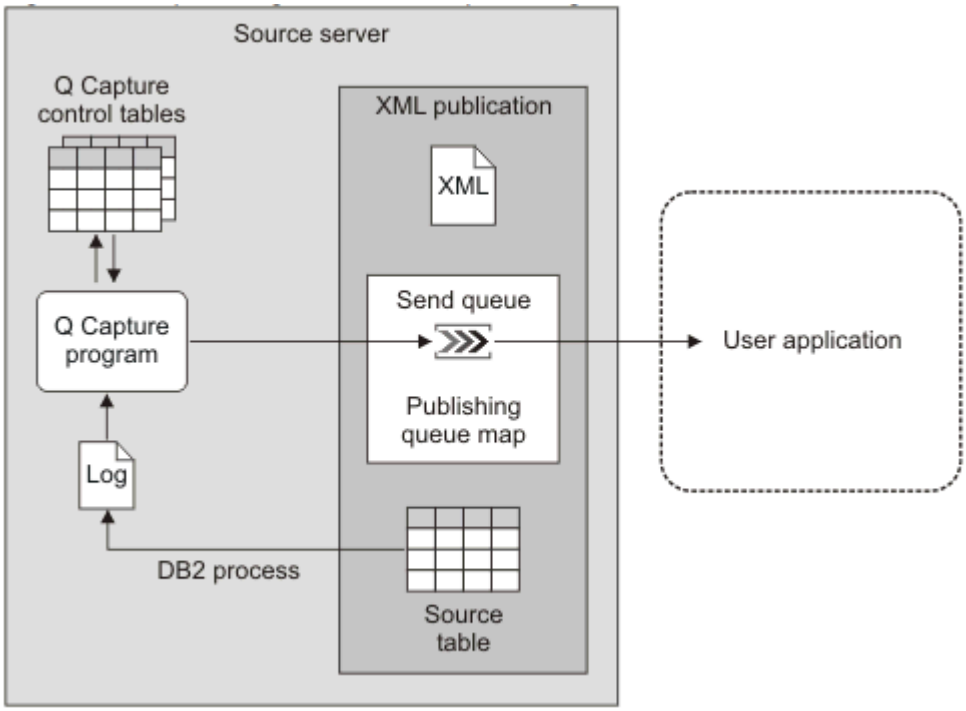

Figure 5: DB2 event publishing (IBMb)

Locking & transaction management

DB2 introduces a number of technologies in order to improve concurrency and lock avoidance. It permits the deferral of row locks for Cursor Stability or Read Stability isolation scans in some situations, until a record is known to satisfy the predicates of a query. The default behavior, when row locking is done during a table scan, is to lock each row before determining whether the row qualifies for the query. It is possible to defer locking after the determination step. Using a DB2 internal variable - *DB2_EVALUNCOMMITED* - predicate evaluation can occur on uncommitted data. It is also possible to improve concurrency by setting the registry variable *DB2_SKIPDELETED* and *DB2_SKIPINSERTED* which permit scans to unconditionally skip uncommitted deletes and inserts, respectively (IBMa).

DB2 provides different isolation levels or locking strategies. Choosing an appropriate isolation level ensures data integrity and also avoids unnecessary

locking. The isolation levels supported by DB2 are (GBC+02):

- Uncommitted Read: it can be used to access uncommitted data changes of other applications, i.e., an application will return all of the matching rows for the query, even if that data is in the process of being modified and may not be committed to the database. Uncommitted read transactions will hold very few locks. Thus they are not likely to wait for other transactions to release locks.
- Cursor Stability: this is the default isolation level and locks any row on which the cursor is positioned during a unit of work. The lock on the row is held until the next row is fetched or the unit of work is terminated. An application using CS cannot read uncommitted data. The application locks the row which was fetched and allows no other application to modify its content. As the application locks only the row on which the cursor is positioned, different results may be delivered to two identical queries.
- Read Stability: all rows are locked that are part of a result set. If a table contains 5000 rows and the query returns five rows, then only five rows are locked. Using this isolation level an application cannot access uncommitted data. Instead of locking a single row, it locks all rows of the result set.
- Repeatable Read: this is the highest isolation level available in DB2. It locks all rows referenced by an application, no matter how large the result set is. An application using RRÂğ cannot read uncommitted data of a concurrent application.

DB2 isolates transactions from each other through the use of locks. This way DB2 controls how other transactions interact with a resource. The DB2 Database Manager uses locks to prohibit transactions from accessing uncommitted data. Once a lock is acquired, it is held until the owning process is finished (using COMMIT or ROLLBACK). To avoid deadlocks the DB2 uses a database manager which monitors locks and lock information. To reduce locks DB2 recommends using the following strategy (IBMa):

- issue the COMMIT command at the right frequency
- choose appropriate isolation level
- specify the FOR FETCH ONLY clause
- use type-2 indexes
- tune the *LOCKLIST* and *MAXLOCKS* configuration parameters.

The introduction of the lock avoidance and CS or RS isolation scans helps to improve concurrency dramatically. It is implemented by setting an appropriate registry variable (*DB2_EVALUNCOMMITED*, *DB2_SKIPDELETED* and *DB2_SKIPINSERTED*). It must be noted, that these registry variable settings apply at compile time for dynamic SQL, and at bind time for static SQL (IBMa).

3.2. Open Source Solutions

This section introduces the most popular Open Source database solutions - MySQL and PostgreSQL. The offerings from MySQL and PostgreSQL have changed dramatically since their introduction in the 90ies. Started as a simple database with a minimal features support, both products grew up to enterprise-level solutions featuring almost all the key-technologies implemented in DB2 and Oracle. It is because of the OS solutions, MySQL and PostgreSQL in the first place, IBM and Oracle were forced to make their software available for public download and even to introduce a free of charge entry level versions of their databases which was hardly imaginable in the past.

This chapter starts with a brief description of both products, investigates the minimum system requirements and analyzes the locking and concurrency support. It also draws a feature-based comparison between both databases. The results of this comparison are summarized in a table at the end of the PostgreSQL section. As the focus of this research paper lies in the

area of database clustering, these approaches are thoroughly examined and compared.

3.2.1. MySQL Cluster

MySQL is one of the most popular Open Source databases today. Although the project started as pure Open Source, it grew up and offers commercial support and products. Quite recently it was acquired by Sun Microsystems and therefore a number of changes may occur in the recent future. Today the community offers a number of products, from simple to enterprise ones. MySQL provides a clustering solution as well and describes it as: "*... the industry's only true real-time database that combines the flexibility of a high availability relational database with the low Total Cost of Ownership of open source. It features a shared-nothing distributed architecture with no single point of failure to assure 99.999% availability, allowing you to meet your most demanding mission-critical application requirements.*."[14]

The online documentation, support and the internet community make it easy to learn, use and deploy MySQL based solutions. Using the OS product brings a lot of advantages, but as the above description states, there are also some drawbacks which are going to be discussed thoroughly below.

MySQL offers a clustering version of its software, distributed under GNU GPL license as well as the commercial versions in the following editions:

- MySQL Cluster Standard Edition
- MySQL Cluster Carrier Grade Edition, additionally to SE version adds some extras such as additional APIs, On-Line Add Node and LDAP Directories support.

In this section we are going to discuss the free community version of MySQL Cluster. As the only difference to the commercial contracts is not the key clustering features used but the online support and a variety of options,

[14] http://www.mysql.com/products/database/cluster/

e.g., custom MySQL cluster builds, consultative support, problem resolution support (via phone, email), custom development, geographical replication, etc[15].

Features overview

As the description of MySQL Cluster states, not all the clustering features are supported. MySQL Cluster implements the key clustering features with the following main differences:

- only shared-nothing architecture
- load balancing using NDB storage engine
- SQL standard compliance is mediocre

MySQL Cluster consists of three kinds of nodes:

1. a data node stores and instantaneously replicates all the data belonging to the cluster. These nodes, also described as slave nodes, manage database transactions.
2. a management server node handles system configurations. Often only one node is needed, as it is essential only on system start-up and system re-configuration. The system remains online regardless of the status of these nodes.
3. a server node enables SQL access to the clustered data nodes. The MySQL server node functions as a MySQL master and handles all the requests. Additional nodes are added to increase performance.

MySQL Cluster implements a typical master-slave architecture. The basic configuration is illustrated in Figure 6 and would typically look as follows:

- one server node,
- one management server node,
- four data nodes for extra performance, capacity and stability.

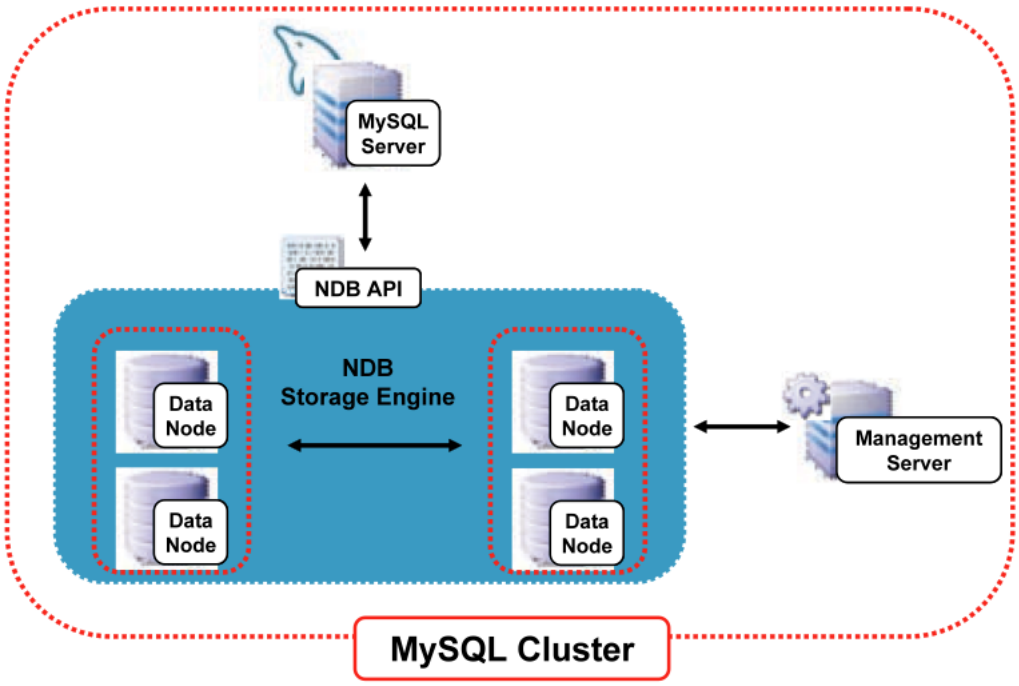

Figure 6: Default MySQL Cluster configuration (Sun)

Adding additional server nodes boosts the cluster performance. Such a configuration is also known as a multi-master configuration. The typical MySQL master/slave replication process is illustrated in Figure 7.

MySQL Server supports Statement Based Replication and Row Based Replication. Asynchronous replication method always uses RBR. The other replication methods are not supported by MySQL Cluster directly, but it could be achieved using third party tools (Sun).

MySQL also comes bundled with a cluster monitoring software and has a number of technologies implemented for performance fine tuning.

Locking & transaction management

External locking and transaction management has the Oracle-like implementation. Each process needs to acquire access to a table before proceeding accessing the table. When access is denied, the process is blocked and kept

[15]http://www.mysql.com/products/database/cluster/support.html

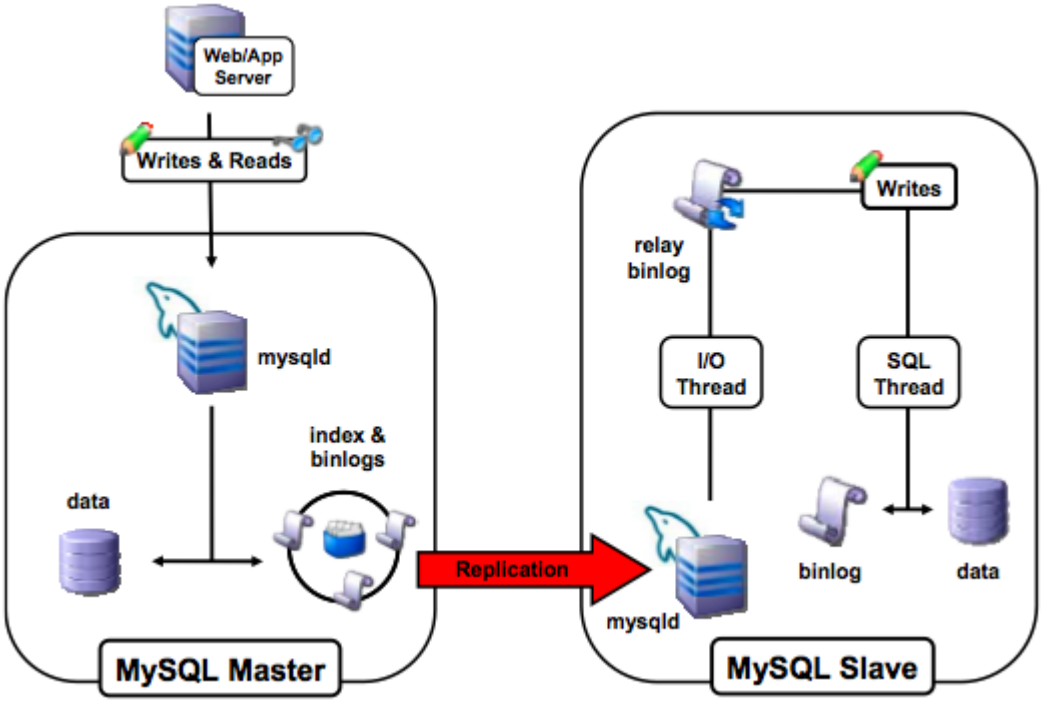

Figure 7: MySQL Cluster replication (Sun)

on hold, waiting until the lock is released by the original process. As already described in the DB2 section, the external locking affects server performance dramatically, as the server must sometimes wait for other processes before it can access the table. In MySQL it is possible to avoid table locking completely by tweaking its internal variables - *skip_external_locking*, thus enabling dirty reads. It is possible to avoid dirty reads as well, by tweaking further MySQL internal parameters, such as *–delay-key-write* and READ COMMITTED isolation level, enabling consistent non-locking reads and thus behaving the same way Oracle does (Sun).

Transaction management was not supported by MySQL until quite recently. Using *InnoDB* MySQL supports the transaction model. Locking is done on the row level and executes queries as non-locking consistent reads by default, in the style of Oracle. Typically several users are allowed to lock every row in tables, or any random subset of the rows, without causing any conflict or memory exhaustion.

3.2.2. PostgreSQL

PostgreSQL is another OS, powerful and rock-solid database system. It has more than 15 years of active development and earned a strong reputation for reliability, data integrity, and correctness. It runs on all major operating systems. It is fully ACID compliant, as well as all other databases reviewed previously, has full support for foreign keys, joins, views, triggers, and stored procedures. It includes most SQL92 and SQL99 data types and it considerably better supports SQL standard than MySQL. PostgreSQL prides itself in standards compliance. Its SQL implementation strongly conforms to the ANSI-SQL 92/99 standards. The extra point worth mentioning is that PostgreSQL has an exceptional documentation[16].

Features overview
Considering the fact that PostgreSQL does not support clustering out-of-the-box, we shall concentrate on reviewing only its main cluster relevant features which have indirect influence on clustering performance.

PostgreSQL shines when it comes to standards compliance, as its implementation strongly conforms to the ANSI-SQL 92/99 standards. It has full support for subqueries (including subselects in the FROM clause), read-committed and serializable transaction isolation levels.

The following non-standard features are implemented and fully supported by PostgreSQL[17]:

- Generalized Search Tree indexing is an advanced system which brings together a wide array of different sorting and searching algorithms including B-tree, B+-tree, R-tree, partial sum trees, ranked B+-trees and many others. It also provides an interface which allows both the creation of custom data types as well as the use of extensible query methods to search them. GiST is also a base framework for many public projects such as OpenFTS and PostGIS. OpenFTS which stands

[16] http://www.postgresql.org/about/
[17] http://www.postgresql.org/about/

for "Open Source full text search engine" is an online engine for data indexing and relevance ranking to support full text search on a database level. PostGIS adds support for geographic objects in PostgreSQL, allowing it to be used as a spatial database for geographic information systems.

- table inheritance puts an object oriented slant on table creation. It allows to derive new table from other ones, the same way inheritance is implemented in object oriented languages, like Java. Even better, PostgreSQL supports both single and multiple inheritance in this manner.
- the rules system, allows the database designer to create rules which identify specific operations for a given table or view, and dynamically transform them into alternate operations when they are processed.
- the events system is an interprocess communication system in which messages and events can be transmitted between clients using the LISTEN and NOTIFY commands. Notifications can be issued from triggers and stored procedures, thus allowing clients to monitor database events such as table updates, inserts, or deletes as they happen.

Describing all the supported features would need a separate white-paper document and therefore is not covered here. The only thing to note is the fact, that based on technologies implemented and supported, PostreSQL is far more advanced than MySQL. The major legal difference between MySQL and PostgreSQL is its license agreement. As stated above MySQL relies on the GPL license[18], PostgreSQL is distributed using BSD license[19]. It makes a huge difference, when it comes to customization and deployment of the commercial products based on this database solutions.

Table 3 provides a brief feature-based comparison between MySQL and PostgreSQL (Ler07):

[18] http://www.mysql.com/products/database/cluster/features.html
[19] http://www.postgresql.org/about/

	MySQL	*PostgreSQL*
License	GPL and Commercial	BSD
Speed	High	Medium
Stability	Very High	High
SQL standard compliance	Medium	High
ACID compliant	yes	yes
Authentication method	SHA1	PAM, LDAP, SSPI, GSS-API, trust, password and Kerberos
SSL support	yes	yes
Locking & Concurrency	high (Oracle-style)	high
Transactions	yes	yes
Replication	yes	yes, relies on 3rd party tools
Load balancing	yes	**NO**
IPv6 support	yes	yes
Clustering	yes	**NO**

Table 3: Brief key-feature based comparison between MySQL and PostgreSQL

Note: When it comes to stability, MySQL is considered to be more stable than PostgreSQL, due to its much larger user base. MySQL source code is better tested and the database is more often used in production environment[20].

Locking & transaction management

PostgreSQL provides a rich set of tools to manage concurrent access to data. Internally, it uses MVCC, i.e., while querying a database each transaction sees a snapshot of data, regardless of the current state of the underlying data. Thus protecting the transaction from viewing inconsistent data that could be caused by concurrent transaction updates on the same data sets, providing transaction isolation for each database session. MVCC minimizes lock contention in order to allow for reasonable performance in multiuser environments (Posc).

[20] http://www.geocities.com/mailsoftware42/db/

The main advantage of using of MVCC rather than locking is that locks acquired for querying (reading) data do not conflict with locks acquired for writing data, and so reading never blocks writing and writing never blocks reading.

Table- and row-level locking are also available in PostgreSQL, however, proper use of MVCC will provide better performance than locks. In addition, application-defined advisory locks provide a mechanism for acquiring locks that are not tied to a single transaction (Posc).

PostgreSQL provides four transaction isolation levels as described in Table 4[21]:

Isolation level	*Dirty Read*	*Non-repeatable Read*	*Phantom read*
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Table 4: PostgreSQL isolation levels

When compared to the Oracle implementation, it is obvious that the isolation levels support in Oracle and PostgreSQL are identical.

The reason for using PostgreSQL instead of MySQL is not only its reliability, but also its support of stored procedures and built-in PglSQL support. It also offers b-tree, hash, r-tree, and its own custom GiST index type which allows for user defined types, and function based index to be created (Ler07). PostgreSQL is rock-solid reliable and has a focus on ACID-correctness: when it returns from a commit, the data is safely on disk and will not be lost.

[21] http://www.postgresql.org/docs/8.3/static/transaction-iso.html

3.3. Summary

The goal of this chapter was to analyze the four most commonly used databases - two commercial solutions represented by Oracle RAC and DB2 Enterprise Edition and two OS alternatives - MySQL and PostgreSQL, bearing in mind their features, clustering approaches and generally to provide a clear picture of the possible database usage. Comparing all four approaches, it is obvious, that Oracle is the industry's golden standard, supporting a wide range of key-technologies and even going far beyond it. Deploying such a solution would guarantee standard compliance, provide a developer with rich set of tools and support, but it comes with a heavy price tag.

MySQL follows Oracle-style philosophy, and with the latest version (>=5.0) strives to be as standard compliant as possible. High stability, speed and outstanding support makes MySQL an excellent choice at no or minimal cost. Providing enterprise features and key-technologies out-of-the-box gives MySQL a big advantage and makes it a worthy alternative.

PostgreSQL is a rock-solid standards compliant product with a rich set of database tools. The lacks of clustering support in the default package per se is somewhat confusing. The cluster implementation is provided by third party frameworks, but it would be nice to see PostgreSQL own cluster implementation in the future. The PostgreSQL Clustering Frameworks and their inter-comparison is thoroughly described in the next chapter of this paper.

Part II.
Application

4. Database clustering in PostgreSQL

This chapter introduces third party clustering frameworks for PostgreSQL. The database does not deliver clustering feature by default and therefore relies on the projects described here. There are a number of frameworks which compensate for the missing built-in technology.

A research of products which provide replication support based on PostgreSQL, revealed two different categories of products: commercial and OS ones. Both categories have different target audiences. Bellow we give a list of available commercial and OS frameworks and their brief description:

1. commercial products – customized clustering solutions for PostgreSQL. Commercial providers concentrate on installation, configuration, support and maintenance of clustering solutions based on PostgreSQL using existing OS frameworks:
   - Mammoth Replicator is 100% pure PostgreSQL with integrated replication, written in C and designed with reliability in mind.
   - Cybercluster provides a synchronous multi-master replication solution for PostgreSQL.
   - Continuent Tungsten Enterprise is an enterprise ready solution for Data Availability and Database Performance Scalability which aims to resolve some of the key points and problems of Oracle, MySQL and PostgreSQL;
2. OS solutions – each reviewed solution is an independent stand-alone project which implements a certain clustering approach:
   - Log Shipping is a core PostgreSQL function which makes a simple replication possible.
   - Postgres-R is a PostgreSQL extension providing consistent database replication.

- PGCluster is the synchronous replication system of the multi-master composition for PostgreSQL.
- PGPpool-II is described as middleware that works between PostgreSQL servers and database clients.
- Slony-I is a master-slave replication system, that includes all features and capabilities needed to replicate large databases.
- Londiste is a Python based replication engine based on the PgQ transport mechanism which is a part of SkyTools.
- Bucardo is a Perl based master-master and master-slave replication system for PostgreSQL.

In the first part of the chapter, I give a general overview of commercial products and then introduce each project separately. The analysis schema for the commercial products differs from the one for the OS solutions. The commercial offerings target primarily users, who would prefer to have their DBMS solutions outsourced, and the main focus lies in the offered services, rather than in used technologies. For the commercial products we give a product overview, describe offered services and stress the differences to other commercial offerings. The OS products, on the other hand, are reviewed from the point of view of a professional user, e.g., a system administrator, who wants to implement a PostgreSQL based clustering solution. The following schema is used for the framework analysis:

- a framework overview – it introduces the framework in question, lists system requirements and underlines key-features.
- clustering approaches – it describes the clustering approaches which the framework uses, compares them to the available ones, described in the previous chapter.
- advantages and drawbacks – it underlines advantages and disadvantage of the used clustering implementations.

The goal of the chapter is to outline the supported clustering features, to compare the available solutions and to choose suitable clustering frameworks for evaluation. The result of this research is a feature-based comparison matrix of chosen clustering solutions. Based on the research, we also try to answer the question which framework(s) or which constellation of frameworks could be used in a production environment.

4.1. Commercial Products

There are a few commercial clustering solutions available for PostgreSQL. Although the products are considered to be commercial, they are distributed under BSD (Mammoth Replicator and Cybercluster) or GNU (Continuent Tungsten Enterprise) license and are generally OSS.

The main focus are not the clustering products per se, but rather offered services, maintenance, on-demand support and personnel training. The companies offering such solutions do not concentrate on providing and describing technical details of their products, as the underlying OS solutions are very well documented, but rather concentrate on providing a wide spectrum of services and on implementing a clustering solution based on specific client needs. The main audience for the commercial providers are clients who use DBMS for their particular needs and prefer to have their database clustering solution to be taken care of by professionals, i.e., outsourced. Therefore the analysis schema differs from the one for OSS. The commercial products are reviewed using the following criteria:

- offered services.
- database migration.
- online support and community.
- used clustering approach and OS framework(s).

4.1.1. Mammoth Replicator

Command Prompt, Inc. describes itself as: "the oldest and largest dedicated PostgreSQL support provider in North America. Since 1997, we have been developing, supporting, deploying and advocating the use of the World's Most Advanced Open Source Database". The company consists of 12 experienced professionals, who develop and support their product. Technologies such as savepoints, shared row locking auto-vacuum integration were introduced by the company's engineers.[22]

The company provides the PostgreSQL Replication engine which is developed exclusively for PostgreSQL with reliability in mind. The product supports the following features (Com):

- it supports master-slave replication.
- it uses a transaction log, asynchronous replication model and is designed to be WAN tolerant.
- replicator supports a number of advanced features, such as large object, role and ACL replication.
- it also supports promotion and failover.
- it implements per table replication.
- it uses SSL and compressed connectivity for greater efficiency, combined with heartbeat and other failover software.

The company provides PostgreSQL professional services which include consulting, case studies, around the clock support (web, phone, remote hand, reactive and proactive), training, project management and infrastructure services. The company actively contributes to the PostgreSQL community, offers custom feature implementations for PostgreSQL, custom development

[22] http://www.commandprompt.com/about/

for PostgreSQL and surrounding applications. It can provide proactive support for demanding environments, such as monitoring and performance measurements. Command Prompt, Inc. also provides commercial support for Slony-I. A number of training courses are offered by the company's professionals as well.

4.1.2. Cybercluster

Cybertec is an Austrian PostgreSQL Database Company which, as it states, offers a comprehensive set of services for PostgreSQL, distributing the clustering solutions for all platforms including Linux, Solaris and Windows. It offers not only commercial support for PostgreSQL, but also provides PostgreSQL training courses and consulting services. Database migration and PostgreSQL core development also belong to the core activities. The company delivers clustering solutions (high-availability, failover and replication) based on the database as well.[23]

The clustering PostgreSQL product from Cybertec is called Cybercluster and is distributed under BSD license. The solution uses synchronous multimaster replication and provides the following advantages (Cyb):

- it is bundled with a replication manager, a load balancer and a connection pooling software.
- it does not use a modified version of PostgreSQL, i.e., a user could use the latest version of PostgreSQL and apply the updates/patches directly.
- it supports the replication over large distances and bad network connections (the product uses Skytools, a software package written by Skype which is design to cope with large server farms. The SkyTools are reviewed later in this paper).
- disaster recovery tools are part of the package.

[23] http://www.postgresql.at/english/produkte_postgresql_e.html

- improved statement based replication (sequences, now(), random(), etc).

Cybertec relies on the OS frameworks to deliver all the core features and implements a number of extended functions, such as: correct handling of non-deterministic functions (nextval / setval, now(), randon(), etc.), no triggers or primary keys are required for replication and additionally it supports DDL events replication.

Reviewing the product we found the following minor gripes:

- the installation packages available for download support only Linux and Windows platforms. Packages for OSX and Solaris have to be compiled from the source code. The windows package is based on the old PostgreSQL revision.
- migration tools are not provided. More than that, the previous PostgreSQL installation has to be removed completely prior to the installation.

These small disadvantages exist only if one wants to test the product himself. Retaining company's services would provide a client with professional technical support (24x7) and training, consulting and performance tuning. The company's services also include cluster design, setup and implementation, including implementation of applications on top of Cybercluster.

4.1.3. Continuent Tungsten Enterprise

Continuent is a fully supported, enterprise ready solution for Data Availability and Database Performance Scalability. It provides clustering solutions for PostgreSQL, MySQL and Oracle. The key features of the products are:[24]

- master/slave and multi/master setups.

[24] http://www.continuent.com/solutions/overview/tungsten-for-postgresql

- automated back-up.
- disaster recovering over WAN.
- replication from Oracle into PostgreSQL.
- failover solution.
- autonomic cluster management
- bi-directional replication

In comparison to the other reviewed commercial solutions, Continuent does not concentrate only on PostgreSQL, but supports Oracle and MySQL as well. It offers a wide range of data availability and performance solutions. On the other hand, Continuent does not offer such a degree of support as the previous companies. Depending on a acquired license, Contiuent offer technical support for its products, such as: professional documentation, quick installation builds, 8x5 email support, 24x7 enterprise support, custom bug fixes and guaranteed response time.

Table 5 sums up the features offered by the reviewed commercial solutions and visually compares them to each other.

	Mammoth Replicator	*Cybercluster*	*Continuent*
Technical criteria			
License	BSD	BSD	GNU
Supported platforms	Unix-type	all platforms	all platforms
Sync. method	Async	Sync	Async
Connection pooling	no	yes	no
Query partitioning	no	yes	yes
Load balancing	no	yes	yes
Monitoring & Administration tools	yes	yes	yes
Latest PostgreSQL support	yes	no	yes
Modified PostgreSQL build	yes	no	no
Commercial criteria			
Training	yes	yes	no
Consulting	yes	yes	yes
Custom implementations	yes	yes	no
Database migration	no	yes	no

Table 5: Commercial PostgreSQL based products

4.2. Open Source Solutions

One of the main reasons for going with a commercial provider for PostgreSQL clustering services is to gain commercial support in the first place. This section introduces the available OS frameworks and describes the advances made in the PostgreSQL clustering. It reviews the frameworks using the same analysis schema. At the end of this chapter a comparison matrix is provided with a summary of the reviewed OS clustering solutions.

Before we start with reviewing the OS products, we have to define the criteria which make a good enterprise clustering product from the technical point

of view, i.e., if a database administrator were to implement a PostgreSQL based clustering solution:

- online documentation (Guides, HOWTOs, FAQs).
- online support and active community.
- simple installation procedure, the modification of the PostgreSQL database is not desirable.
- the distribution should support the latest version of PostgreSQL.
- connection pooling, load balancing, query partitioning and caching features is a plus.
- monitoring and administration software.

4.2.1. Log shipping

Write Ahead Log (WAL) is an implementation provided by PostgreSQL itself, the changes of each transaction are written to the DB after these changes have been logged in a write ahead log file. The changes from WAL are then read and applied to other databases. Such an approach results in a reduced number of disk writes, as only the log file needs to be flushed to disk to guarantee that a transaction is committed. WAL makes it possible to support on-line backups and point-in-time recovery.[25]

Such an approach may have some drawbacks which depend on the application:

- slave nodes cannot serve requests during replication.
- slave nodes cannot be replicated while online.
- PostgreSQL natively accepts only completed WAL files.

[25] http://www.postgresql.org/docs/8.3/interactive/wal.html

WAL Replication is very similar to the mechanism PostgreSQL uses when recovering from an unclean shutdown. The PostgreSQL daemon could not determine the database state, it opens WAL and replays all the transactions since the last checkpoint.

PostgreSQL has a configurable parameter command which is called every time after a segment switch, thus making it easy to deliver complete WAL segments from a master server to slaves, e.g., using scp, rsync, etc. When replaying WAL segments, the PostgreSQL database must be brought offline. When recovery is over, the database in question is taken back online.

The Log Shipping works as advertised, it is a part of PostgreSQL and is very well documented. It is a good solution if one wants to have an up-to-date backup copy of a database. But such an approach lastly can not be used for heavily loaded clusters, as the slaves would then be offline the majority of the time and secondly it provides neither a load balancer nor a connection pooler. Besides, bringing a slave database server offline during the replication is not exactly the best alternative. From the performance and high availability point of view, relying on such an approach for database replication is at least questionable.

4.2.2. Postgres-R

Postgres-R is an extension to the PostgreSQL, providing efficient, fast and consistent database replication for clusters. It is designed to be as transparent as possible to the client, stable and secure by default. The primary use of Postgres-R is to build a load-balancing and high-availability database systems. Due to the flexible architecture of Postgres-R, it is easily possible to extend or adjust the replication process to many different means. Compared to common single node database systems, a Postgres-R cluster is more reliable and scales better while being cheaper and more flexible.[26]

[26] http://www.postgres-r.org/about/

Postgres-R is an effort to integrate a replication solution into PostgreSQL. It is not a part of PostgreSQL, as it is not a mature and well tested solution, i.e., it did not reach production quality as of yet (Posa).

Porsgres-R relies on the shared-nothing synchronous cluster architecture, it provides eager replication, supports multi-master configuration on the basis of binary change-set replication. The framework consists of a replication manager which has a cluster coordination function. The transaction management is handled by each node separately. In the case of a conflict, all the nodes participating in the replication cluster execute a roll-back. In the other case a successful commit is issued - a so called two-phase commit and the data becomes a part of the change-set (Posa).

Postgres-R is a separate project, thus a user may have the latest PostgreSQL database installed on his system.

There are a couple of disadvantages when using Postgres-R:

- an experimental code base, the project is still in development and not ready for production environment as of yet.
- the documentation is still lacking, no Guides, HOWTOs or FAQs are available online.

The project meets all the other criteria defined for a good PostgreSQL clustering solution. Though, it is under heavy development, it promises to be a very reasonable alternative. And if it is merged with the PostgreSQL code base, it is going to provide a clustering feature to PostgreSQL out-of-the-box.

4.2.3. PGCluster

PGCluster is the synchronous replication system of the multi-master composition for PostgreSQL. PGCluster uses a modified version of PostgreSQL and consists of (PGC):

- a load balancer.

- a cluster DB.
- a replication server.

There is no other information available online. The project looks dead, there was no update since March 2008. The home page[27] produces a list of Python errors all the time. The previous release took place in 2005, the project activity is very low. So PGCluster fails to provide an acceptable clustering framework for PostgreSQL as well.

4.2.4. Londiste

Londiste is a part of SkyTools developed by Skype. The project is a replication engine written in Python. It uses PgQ as a transport mechanism. Its main goals are robustness and easy usage. Thus it is not as complete and feature-full as Slony-I.[28]

Londiste is an asynchronous master-slave replication system. Asynchronous means that a transaction committed on the master is not guaranteed to have made it to any slave at the master's commit time. Data changes on slaves are not reported back to the master, it is the other way around only.[29] Londiste is a part of the SkyTools package and can be easily installed using a package manager on any Linux distribution. Although it is not as flexible or feature-full as PGPool, it is a very light-weight solution which can provide some sound and acceptable clustering solution. The only point of concern here is, that it still uses asynchronous synchronization which is not always acceptable, e.g., when data consistency is an issue.

4.2.5. Bucardo

Bucardo is an asynchronous master-master and master-slave replication system for PostgreSQL written in Perl. It uses triggers on individual tables.

[27] http://www.pgcluster.org/pgcluster
[28] https://developer.skype.com/SkypeGarage/DbProjects/SkyTools
[29] http://skytools.projects.postgresql.org/doc/londiste.cmdline.html

It supports conflict resolution and exception handling through the use of custom Perl subroutines.[30]

Bucardo does not require PostgreSQL modifications. As the framework is written in Perl, it requires that Perl and a number of modules are installed on the system. The software has only been tested on Linux and BSD platforms and will not run on Windows without modifications to the source code.

The supported features of the framework are (Buc):

- it uses fast, asynchronous trigger-based replication, both master to slave, and master to master.
- it supports standard and custom conflict handling methods.
- it provides custom exception handling methods and other hooks for fine control of the replication process.
- it implements graceful handling of network disconnections and other problems.
- it is easy to configure and setup.
- it supports rewrite of target tables with custom SELECT clauses.
- step by step data changes are not tracked, so updates can happen quicker.
- it includes logging and analysis tools.

The authors of the project also list a number of the known limitations:

- it replicates tables only, not the entire database.
- it does not replicate DDL.
- it cannot handle more than two master nodes at a time.
- it also requires a primary key on each table to be replicated.

[30] http://bucardo.org/bucardo.html

- data changes are not tracked, thus expensive locking could not be avoided.

Besides of the known limitations mentioned above, it lacks an online community, runs only on couple of platforms (in some cases even the source code modification are needed), requires an outdated Perl installation and its libraries, no built-in load balancer or connection pooler and the list of the required but not implemented features goes further. If an asynchronous replication is acceptable, a better solution would be to go for Londiste, rather than using Bucardo.

4.2.6. PGPpol-II

Pgpool-II is a very interesting project. It is a middleware that works between PostgreSQL servers and PostgreSQL database clients. The framework runs on most UNIX-like platforms, including Linux, Solaris and BSD. The features of PGPool-II are:[31]

- connection pooling is used to reduce connection overhead and to improve system performance by reusing open connections.
- replication is meant to create realtime backups on physical disks in order to provide non-stop servicing.
- load balancing reduces the load on each PostgreSQL server by distributing SELECT queries among multiple servers.
- it uses parallel querying, the requests are divided and executed on all servers concurrently which reduces overall query execution time.
- it supports limiting of exceeding connections which limits the number of concurrent connections. The extra connections are not dropped but rather queued instead.

PGPool has some known limitation, such as (Posb):

[31] http://pgpool.projects.postgresql.org/pgpool-II/doc/pgpool-en.html

- limited authentication methods support: in the master-slave or replication mode it supports trust, clear text and PAM authentication methods. In the other modes supported authentication methods are besides the ones mention in the 1st points are crypt and MD5.
- PGPool does not support access control in pg_hba.conf format. If TCP/IP connections are enabled, all the connections from any hosts are supported. Additional access control could be initiated using the iptables package.

PGPool is distributed under the BSD license, uses synchronous replication and meets the majority of the criteria defined: the installation is straightforward, the online documentation is complete. The solution is reasonable and sound. The known limitations could be avoided or tweaked using some other Linux tools. In this respect, the project looks very promising. The thorough testing and benchmarking of the framework is carried out in the next chapter of this research paper.

4.2.7. Slony-I

Slony-I is another interesting clustering alternative for PostreSQL. Slony-I is a master-slave asynchronous replication system supporting cascading and failover. It includes all features and capabilities needed to replicate large databases to a reasonably limited number of slave systems. Originally the system was designed for use at data centers and backup sites, where the normal mode of operation is that all nodes are available. Slony-I is powerful, trigger based, and highly configurable.[32]

Slony-I is not tight to a particular version of PostgreSQL, i.e., the system could be stopped and started at any time. The framework has though a number of serious limitations (Slo):

- it has no built-in connection pooling, load balancing or query partitioning.

[32] http://www.slony.info/documentation/slonylistenercosts.html

- it is unreliable on unstable network connections.
- it cannot detect a node failure, thus no automatic master node reelection possible.
- a multi-master architecture is not supported.

Slave nodes are used for safety or for parallelizing queries which improves performance. Slony-I suffers from $O(N^2)$ communication overhead, where N stands for the cluster size. There is no failover system delivered with Slony. The project sees it not as a part of clustering solution but relies on a database/system administrator instead.

Connection pooling and load balancing could be implemented using some third party software, e.g., using the SkyTools project. It is hardly possible to use Slony as replication framework with remote database nodes. Although regarding communications, the replication cascading is one possibility to reduce load on the master in a large replication cluster.

Slony does not try to implement all possible replication models, it rather concentrates on implementing one - asynchronous replication, using triggers to collect table updates. It provides an extensive documentation, with Guides, FAQs and Best Practices sections. It comes bundled with monitoring and administration tools.

4.3. SkyTools

Accept for PGPool, the clustering solutions presented in this paper lack a connection pooler, load balancer and query partitioner. Which is a huge disadvantage when a complex clustering approach is required. Fortunately, Skype uses internally PostgreSQL databases as well and therefore it has developed a handy set of database tools – SkyTools. The database access is implemented through stored procedures which has a number of positive effects (Skyb):

- table organization and optimization is transparent for applications.

- easier security management is provided. In most cases it is needed to control user's behavior rather than the tables.
- all transaction can be made in auto-commit mode, thus taking minimal amount of roundtrips for each query and uses less CPU for each transaction.

The SkyTools introduce a set of instruments which are described bellow in full details. In the next chapter we make use of these tools, in order to implement a clustering solution with PostgreSQL using Slony-I:

- pgBouncer is a lightweight and robust connection pooler for PostgreSQL. It reduces the number of open connections to a database, thousands of incoming connections are reduces to only tens, thus saving system resources. Figure 8 schematically illustrates this process.

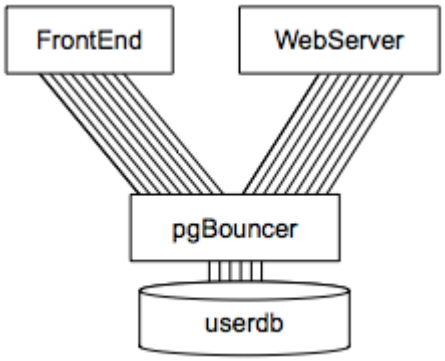

Figure 8: PostgreSQL Connection Pooler (Oja08)

  pgBouncer supports a number of pooling modes:

  - *session pooling* - assigns a connection from the pool to a client and puts back the connection to the pool after the client's disconnect.
  - *transaction pooling* - connections are assigned only during transactions.

  - *statement pooling* - the same as transaction pooling, but it gets a connection only for one statement, multi-statement transactions are prohibited.

- plProxy is a compact language of remote calls between PostgreSQL databases. Using plProxy a user can create proxy functions to be executed on a remote database server (see Figure 9).

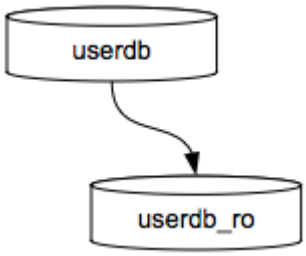

Figure 9: Remote Call using plProxy (Oja08)

The body of the function describes connection parameters. Using remote calls is not a very good idea for data modification as there is no guarantee that the transaction is committed (a two phase commit is not supported), but it could be used efficiently to serve read requests.

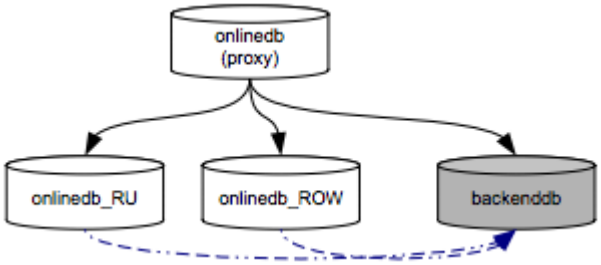

Figure 10: Geographical partitioning with plProxy (Oja08)

plProxy is used as an additional layer between a database and an application. It is in some respect a load-balancer and a security layer, e.g., it can be used for geographical (see Figure 10) or application driven

(see Figure 11) partitioning, to split database based on country code and/or based on a given application context.

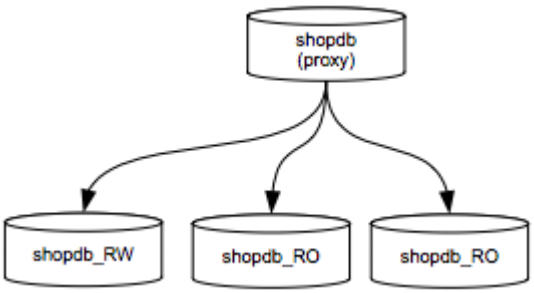

Figure 11: Application based partitioning with plProxy (Oja08)

Horizontal partitioning for load balancing is another area of plProxy usage. As proxy databases are stateless, using plProxy in a cluster for high availability and load balancing is a good idea (see Figure 12)

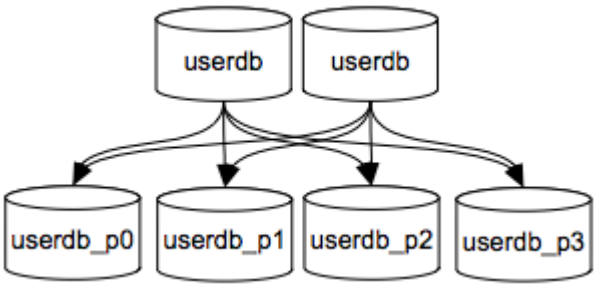

Figure 12: Load Balancing with plProxy (Oja08)

4.4. Summary

In this chapter the available clustering solutions for PostgreSQL were reviewed, commercial and OS ones. The commercial solutions focus on providing a wide range of services (cluster design and setup, consulting, maintenance, on-demand support and training course) bearing in mind specific client's needs. There is a number of alternative OS projects available which

target professional users. Postgres-R looks very promising, but is still in development and not ready for productive use at this very time.[33] We have chosen three solutions which seem reasonable for implementing clustering with PostgreSQL - Slony-I, PGPool and Londiste. The following table summarizes and provides the brief overview of the key-technologies and features used in these frameworks.

	Londiste	*PGPool-II*	*Slony-I*
License	BSD	BSD	BSD
Maturity	stable	stable	stable
Replication method	multi-slave, shared-nothing	master-slave, shared-nothing	single master only to slave, shared-nothing
Sync. method	Async	Sync	Async
Connection pooling	no	yes	no
Query partitioning	no	yes	no
Load balancing	no	yes	no
Online docs & community support	yes	yes	yes
Monitoring & Administration tools	yes	yes	yes
Latest PostgreSQL support	yes	yes	yes

MySQL Cluster is being advertised a lot and often compared to PostgreSQL. The main concern here though is that it writes asynchronously which on the one hand results in a considerable performance boost and better scalability, but on the other hand does not guarantee that the data is on-disk safe and consistent. Bearing in mind that the MySQL Cluster uses asynchronous writes, splitting read/writes operations between the master and the slaves causes inconsistency when, e.g., the data is written and read immediately. There is no way to use synchronous replication with MySQL Cluster itself, but to go with other OS solutions.

[33] http://www.postgres-r.org/documentation/installation

5. Evaluation

In the previous chapters of the paper we provided some theoretical background on clustering, reviewed the most popular databases, looked at the advantages and drawbacks of the implemented clustering approaches and finally introduced the available clustering solutions for PostgreSQL. The database relies on external frameworks for implementing replication. Although it has a core function which may be used for replication, e.g., write-ahead logging, we came to the conclusion, that it has some limitations. Carefully analyzing the frameworks in question based on the selected criteria, we concluded, that three projects are most suitable for building a clustering solution using PostgreSQL: PGPool-II, Slony-I and Londiste.

Slony-I has a couple of drawbacks, in order to compensate for the lack of functionality we are going to use an additional package - SkyTools. PGBouncer and plProxy are used for connection pooling and load balancing respectively. PGPool-II on the other hand comes bundled with its own set of tools and does not require any additional installations.

In this chapter we are going to design a simple cluster, set-up and configure the cluster using these projects and finally run a couple of tests. We are going to evaluate each solution separately, taking into account the goal each project is set to achieve.

5.1. Test cases design

In order to test the selected projects, we are going to design a common cluster configuration, specifying all the requirements to be met by each framework. Figure 13 schematically demonstrates the cluster setup.

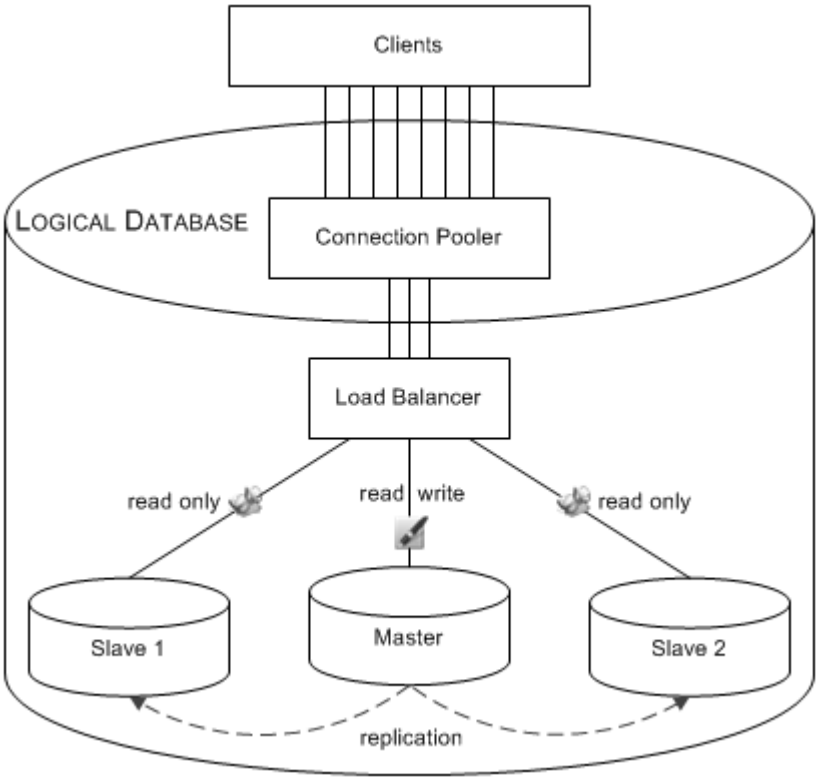

Figure 13: Cluster Configuration

5.1.1. Cluster architecture

For clients, the cluster is just one database instance, transparently represented by a logical database, which consists of:

- a connection pooler which reuses opened connections to the cluster, thus limiting the used resources.
- a load balancer which adjusts the load between database servers. Read only queries are sent to the slaves and modification queries are sent only to the master node.
- a master server which processes the modification queries.
- two slave servers which receive replicated data from the master server and process read only queries.

The idea here is the following: the connection pooler controls all the incoming connections. The number of opened connections in the pool is limited to ten clients. If all the opened connections are used, a new incoming client connection is put in a waiting queue. As soon as one connection in the pool gets released, it automatically gets assigned to the first client in the queue. The load balancer processes the incoming queries and sends all the modification queries to the master server, while the read only queries are balanced among the slaves. Each slave has an equal priority.

Master and slave nodes are installed on a separate Linux machine, which are connected via network. The connection and all the command executions are carried out against the logical database and not against separate database nodes. Thus for a client (independent if it is a command line interpreter or a JDBC client) it seems that a connection is opened to a single PostgreSQL database.

5.1.2. Database schema

Figure 14 graphically illustrates the database schema (the optional attributes are omitted) used for the evaluation.

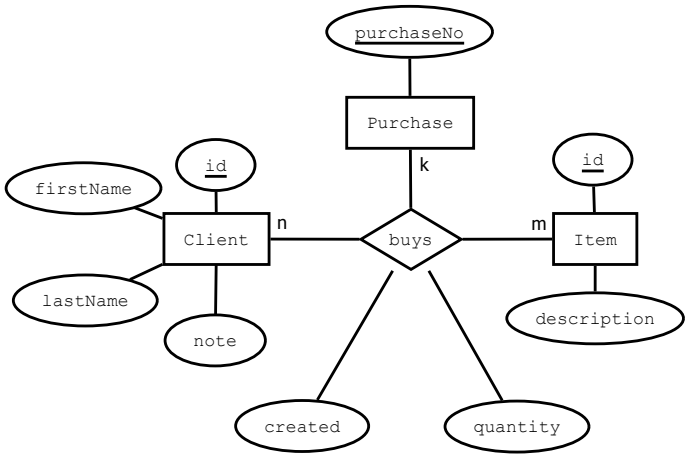

Figure 14: Database schema

We define two entities – a client and an item. Both entities stand in a n to m relationship which results in a separate client_item relation. Each entity (Client and Item) has an *id* as its primary key which is an auto generated and auto incremented value. In MySQL there is a possibility to define an auto incremented column, PostgreSQL uses sequences for this purpose. The SQL schema representation of the test database is given bellow:

```
CREATE TABLE Client(
    id              SERIAL PRIMARY KEY,
    firstName       VARCHAR(50) NOT NULL,
    lastName        VARCHAR(50) NOT NULL,
    note            VARCHAR(255) NOT NUL,
    created         TIMESTAMP,
    lastModified    TIMESTAMP DEFAULT now()
);

CREATE TABLE Item(
    id              SERIAL PRIMARY KEY,
    description     VARCHAR(255) NOT NULL,
    created         TIMESTAMP,
    lastModified    TIMESTAMP DEFAULT now()
);

CREATE TABLE client_item(
    purchaseNo      FLOAT DEFAULT random() NOT NULL,
    clientId        INTEGER REFERENCES Client(id),
    itemId          INTEGER REFERENCES Item(id),
    created         TIMESTAMP DEFAULT now() NOT NULL,
    quantity        BIGINT NOT NULL,
    PRIMARY KEY(purchaseNo)
);
```

Listing 1: dbSchema.sql

The use of non-deterministic functions is not a coincidence, but is one of the main objectives of the research. It is important to know how the chosen clustering frameworks handle these commands. Additionally, it is also interesting to test, how the entities' dependencies are processed which are implemented by the use of the foreign keys.

5.1.3. PGPool-II Configuration

Figure 15 schematically represents the clustering solution using the PGPool-II framework.

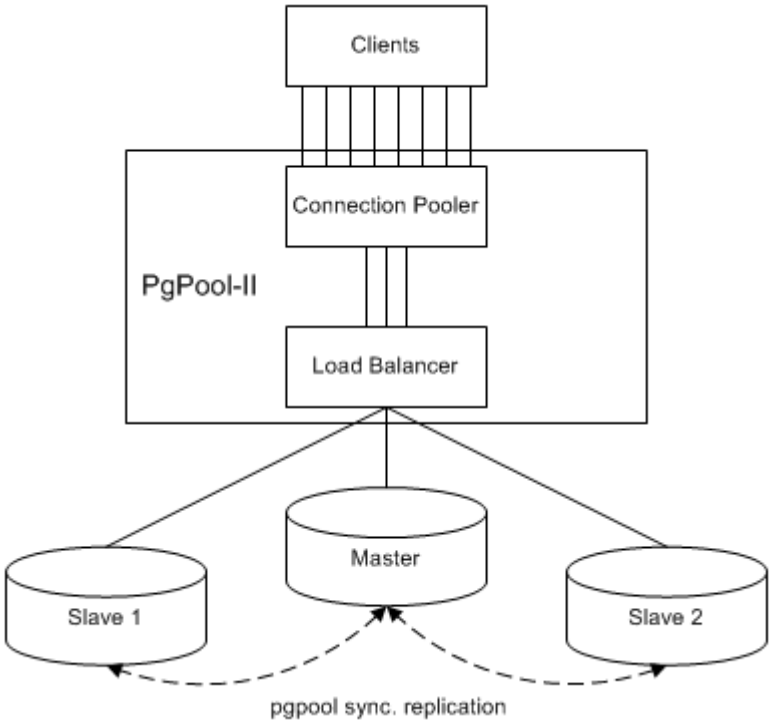

Figure 15: PGPool-II configuration

As already described in the previous chapter, PGPool comes bundled with a built-in connection pooler, load balancer and query partitioner, so no additional software is needed. The detailed PGPool configuration is given in the Appendix A.

5.1.4. Slony-I Configuration

Slony, however, provides neither a connection pooler nor a load balancer. The project concentrates solely on database replication. Therefore, we have chosen to test the following configurations:

1. We bundle Slony-I with PGPool-II. The PGPool framework is used only for connection pooling and load balancing. The database replication is done via Slony. Figure 16 illustrates the proposed solution. The detailed configuration for both software components is found in the Appendix B.

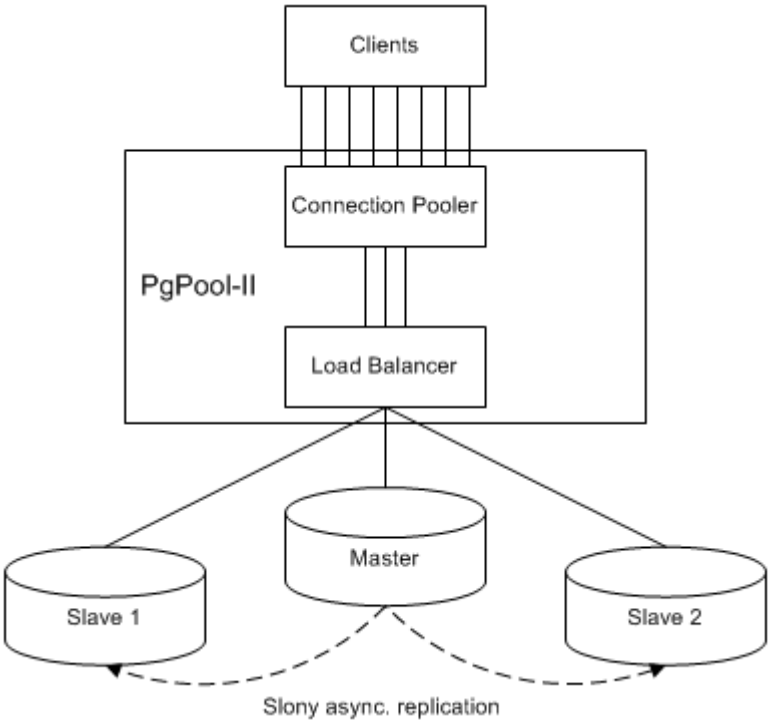

Figure 16: Slony configuration with PGPool

2. Another approach is to use SkyTools (pgBouncer for connection pooling and plProxy for load balancing) instead of PGPool. Figure 17 illustrates the idea.

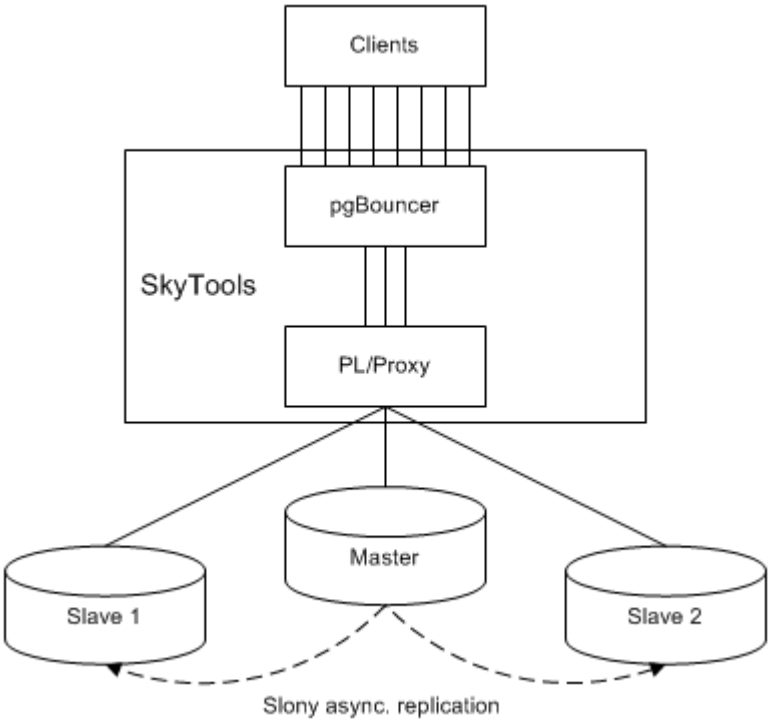

Figure 17: Slony configuration with SkyTools

The point of using two different frameworks for load balancing and connection pooling is to see, whether it is possible to gain some performance advantage by using SkyTools over PGPool. PGPool relies solely on session based pooling, at the same time, it is possible to tweak SkyTool into using a transaction or a statement based pooling which might bring an additional performance boost to the cluster.

5.1.5. Londiste

Londiste comes bundled with SkyTools, thus it is possible to design a clustering solution based only on the SkyTools packages (see Figure 18).

Both Londiste and Slony implement asynchronous replication, so it is possible to draw a direct performance comparison between these two projects.

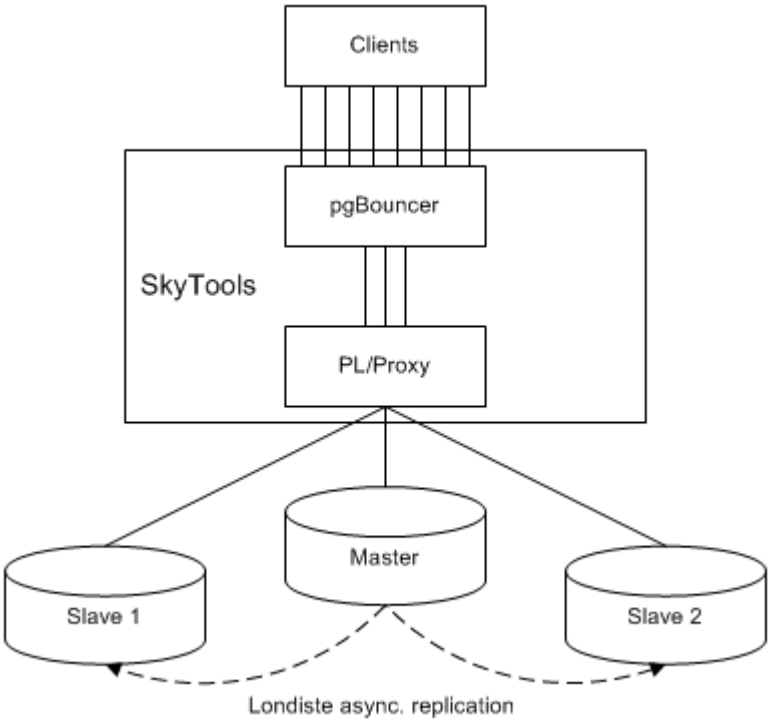

Figure 18: Londiste configuration

The Londiste (the same pgBouncer and plProxy configurations is used here) configuration is found in the Appendix C.

5.1.6. Test cases

As a second step, test cases of different complexity and benchmarking criteria have to be defined. The following test cases are defined to be run with each of the settings described above:

1. *Write* – creates new records for each entity.
2. *Modify* – updates all created records.
3. *Read/Write* – combined read and write operations.

It has to be noted, that all the operations are carried out against the cluster. Clients open connections to the logical database interface and not to the database node(s). Thus all the database schema changes are executed on all the nodes in the cluster if a framework in question supports it (e.g., when PGPool is used, manual schema adjustments have to be carried out otherwise). Additionally, all the test cases are run against a single PostgreSQL instance for performance comparison.

The test cases have the following configuration:

- a number of transactions per client – 1.000 (one thousand)
- a number of concurrent clients – one, five and ten respectively.
- all the nodes in the cluster have an equal priority.
- the connection pool size is set to five.

There were a couple of options how to run the tests against the cluster: whether to use an existing benchmarking tool (pgPerformer[34]) or to develop a new one. The author of this paper decided to develop a new one for the following reasons:

- not all of the clustering approaches replicate the changes to the database schema, thus it would require to create all the tables required by pgPerformer separately on each node, for each test case and for each tested framework.
- there is no macro language in pgPerformer, i.e, it is not flexible in the case that some specific function is required.

The program was designed using Java programming language. In order to achieve some abstraction level, the author used the SpringFramework [35] which provides a JdbcTemplate for database operations and XML beans definitions for beans wiring. The complete source code of the project is found at https://svn.semanticlab.net/svn/oss/thesis/pgcluster.

[34] http://www.semanticlab.net/index.php/Pgperformer
[35] http://www.springframework.org

Each test case is repeated three times, the average execution time and the standard deviation are calculated. The test cases schema with the test result for a single PostgreSQL instance is provided in Table 6.

Test performed	*Concurrent clients*	*Avg. exec time* (t_1)	*TX per sec* (t_2)	σ (t_2)
write	1	2.958	338	3.08
	5	9.119	548	5.51
	10	16.949	590	4.51
modify	1	631	1.585	2.00
	5	2.534	1.973	7.00
	10	5.094	1.963	2.08
read/write	1	3.885	514	8.54
	5	11.022	906	5.77
	10	20.780	962	2.52

Table 6: PostgreSQL single instance - test results

Figure 19 graphically illustrates the performance of a single PostgreSQL instance.

It is also interesting to investigate the performance penalty a clustering solution introduces to a project. Therefore the performance of a single PostgreSQL instance was included as a base line in the test setting.

5.2. Test case 1 - PGPool

Setting up PGPool is easy and self explanatory. The installation on a Ubuntu machine is straightforward using the *apt-get* package manager. It provides a user with a complete pre-configured package and comes with a rich documentation. The framework's web site is a great resource which provides all the needed documentation and has a community forum. The community support is outstanding. The configuration files are self explanatory, it took no time to configure the cluster defined for this paper.

Table 7 provides the tests results:

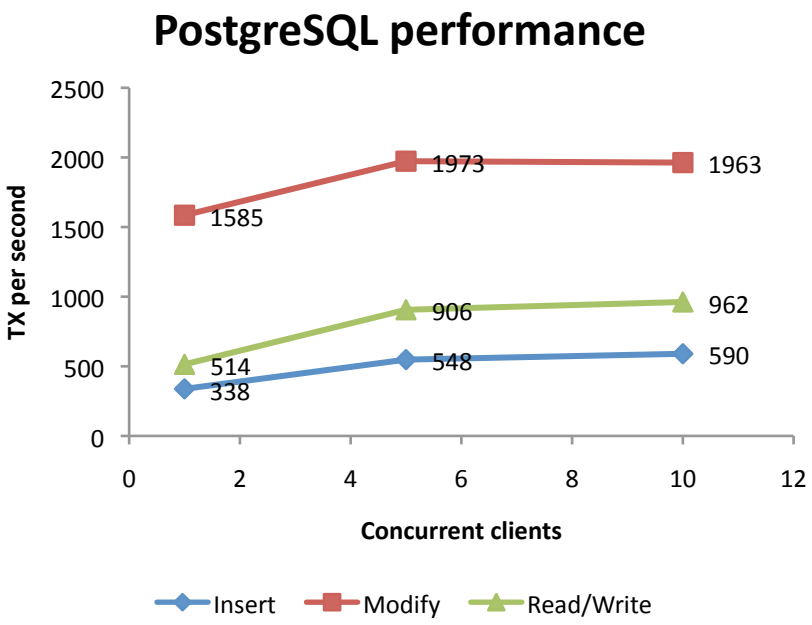

Figure 19: PostgreSQL performance

Test performed	*Concurrent clients*	*Avg. exec time* (t_1)	*TX per sec* (t_2)	σ (t_2)
write	1	23.810	42	3.39
	5	62.500	80	4.10
	10	99.010	101	2.59
modify	1	4.762	210	4.01
	5	17.241	290	5.49
	10	32.154	311	5.39
read/write	1	22.221	90	4.49
	5	72.993	137	5.68
	10	129.870	154	3.48

Table 7: PGPool - test results

The PGPool performance is visualized in Figure 20.

Obviously, synchronous replication is a very expensive. Judging from the results above, PGPool is at least five times slower than a single PostgreSQL

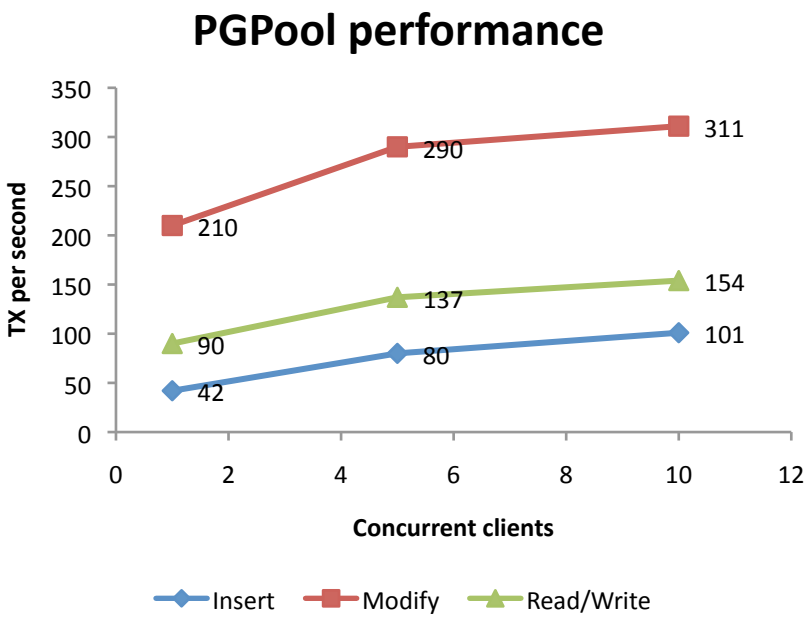

Figure 20: PGPool performance

instance. Therefore such a replication strategy makes sense only, if data integrity and consistency have a high priority. It also makes sense to implement such a replication within the local network, where the network availability could be guaranteed.

PGPool has also a couple of serious drawbacks which came up during the tests:

- it does not handle non-deterministic functions. Such commands as *now()* or *random()* are executed on each server, producing independent and inconsistent results.
- the sequences are treated independently by each server as well.
- session wise load balancing, i.e., all the read-only requests are processed by one node and not balanced between the nodes in the cluster. It is not possible to define a transaction or a statement wise load balancing. It results in a poor read-only performance.

Considering the problems listed above, it is hardly imaginable using PGPool as it is. There are two possible solutions to the raised issues: to use some intermediate layer which intercepts, executes and substitutes non-deterministic functions. Another possible solution would be to adjust the application code itself and to use non-deterministic functions built-in the application programming language. In any way, independent which approach is chosen, it requires some extra work in order to avoid the problems listed above.

5.3. Test case 2 - Slony

Installing and setting up Slony for data replication is a way more complicated than PGPool and requires quite a bit of configuration. The project web-site provides extended documentation and outstanding support. Although a learning curve exists, it helps to understand the underlying technology and the flexibility of this clustering approach. The Ubuntu package comes completely pre-configured. Although, the authors of the project consider load balancing and connection pooling not the part of the project, it would be a welcomed feature. Therefore, we are going to use PGPool and SkyTools.

The Slony cluster configuration is found in Appendix B. In contrast to PGPool, a cluster namespace has to be defined in the first place. At initialization time of every node, Slony creates its own special configuration tables, where the configuration and replication state information is stored. Each node is identified by a number, so some cluster planning is required for complex configurations in advance, as nodes renumbering is not possible at a later stage. The following requirements have to be met for Slony replication:

- Slony-I needs to have primary keys configured. If a table does not have a primary key, Slony will create it.
- a list of tables is required, i.e., replication sets should be defined.

- sequences to be replicated have to be defined in advance too.

Running the write, modify and combined read-write tests produced the results represented in Table 8:

Test performed	*Concurrent clients*	*Avg. exec time* (t_1)	*TX per sec* (t_2)	σ (t_2)
write	1	6.211	161	4.11
	5	16.393	305	6.75
	10	25.907	386	5.70
modify	1	1.527	655	4.93
	5	6.258	799	5.24
	10	10.718	933	4.58
read/write	1	6.173	324	6.18
	5	17.544	570	5.38
	10	30.817	549	4.48

Table 8: Slony - test results

Figure 21 graphically represents the performance of Slony.

It is clear, that using asynchronous replication delivers a significant performance boost in comparison to synchronous replication. During the tests, PGPool was substituted with SkyTools in order to analyze the performance differences. In general, the performance of these frameworks, when it comes to connection pooling and load balancing, seems to be comparable. Testing Slony with PGPool and SkyTools did not deliver any significant performance differences.

An obvious advantage of using Slony over PGPool replication is that it does not execute non-deterministic functions on each node, but rather copies the complete database entry. The other plus, which speaks for using Slony, despite of the configuration and planning overhead, is, that it is quite flexible and a user may define any number of replication sets and rules.

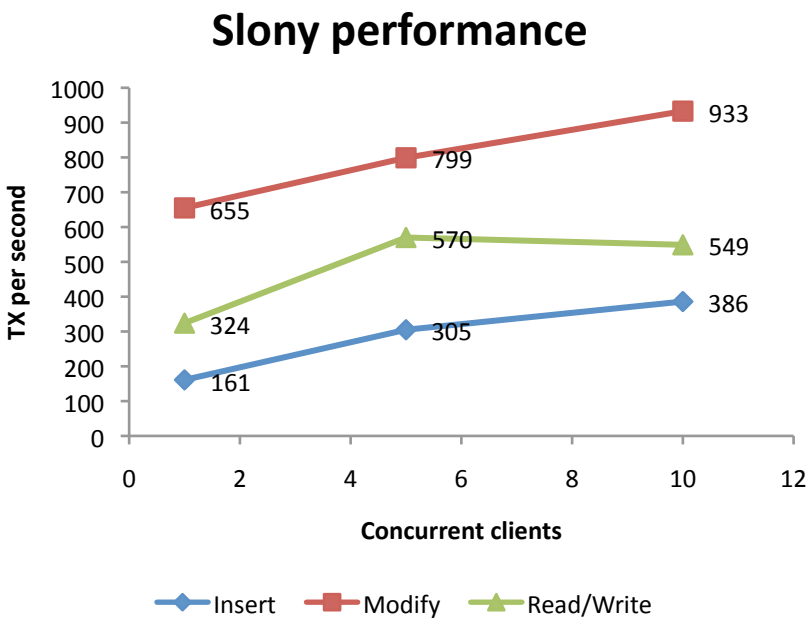

Figure 21: Slony performance

5.4. Test case 3 - Londiste

In comparison to PGPool and Slony, setting up Londiste 2.1.8 under Ubuntu 9.04 (see Appendix C for configuration files) is not easy, despite of the provided documentation. Installing SkyTools requires some tweaking as the package maintainer compiled the package only for Python 2.5, therefore some knowledge or at least understanding of Python is necessary. First of all, it does not work with any version of Python, but 2.5. For test purposes Python was downgraded to the version 2.5. The problems listed above are not connected with SkyTools directly, but are rather the problems of the Ubuntu package.

The asynchronous replication with Londiste produced the results represented in Table 9:

Test performed	*Concurrent clients*	*Avg. exec time* (t_1)	*TX per sec* (t_2)	σ (t_2)
write	1	4.405	227	3.42
	5	15.625	320	5.01
	10	24.938	401	4.60
modify	1	1.490	671	4.30
	5	6.211	805	4.91
	10	9.515	1051	3.01
read/write	1	5.587	358	3.90
	5	16.863	593	3.00
	10	28.289	707	2.34

Table 9: Londiste - test results

Figure 22 illustrates the performance of the Londiste framework.

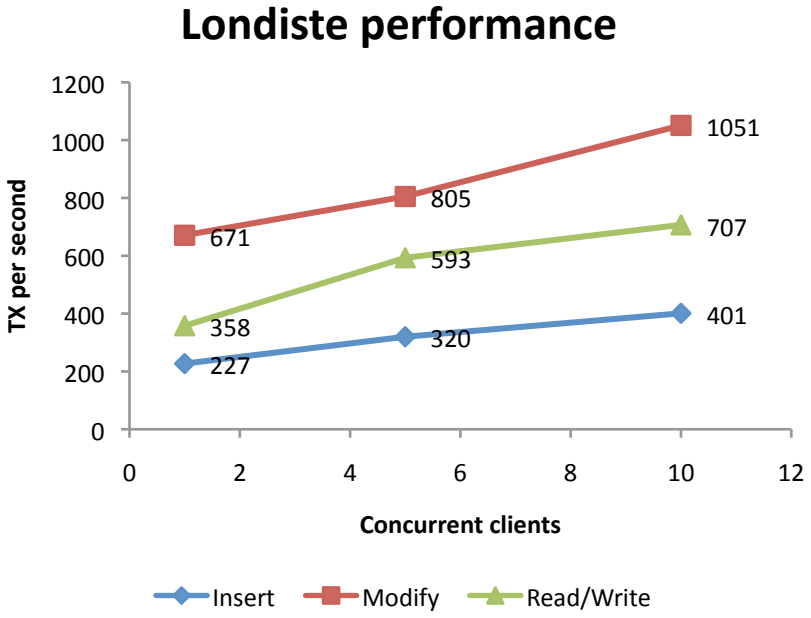

Figure 22: Londiste performance

Compared to Slony, there are some minor performance differences, especially when it comes to record modifications. During the testing the following particularities were found out:

- Londiste does handle non-deterministic functions and copies the entries to the subscribed nodes, rather than executing non-deterministic functions on each dependent slave.
- PgQ guarantees that the changes made to the master will be replicated in exact order to the subscribed nodes, as soon as they become available.
- the changes to a database schema are not replicated automatically, but have to be executed separately on each node in the cluster.
- it is also not possible to replicate the whole database, but each table has to be subscribed explicitly.

5.5. Evaluation

The evaluation section summarizes the three reviewed approaches, investigates the major difficulties and evaluates the tested clustering solutions. It is rather impossible to compare these clustering approaches directly: PGPool uses synchronous replication which is much slower than asynchronous one. It does not mean that it is a bad approach per se. There are some cases when synchronous replication makes a lot of sense and is required, despite of the speed penalty, e.g., when data integrity is mission critical. On the other hand, it is not always practical to use synchronous replication as well. When databases are divided geographically it is more advisable to use asynchronous over synchronous replication, otherwise the cluster performance would be the system's bottleneck. Therefore only Slony and Londiste were compared directly to each other. Slony is a mature feature-full framework and Londiste, being a part of SkyTools project, is a collection of Python scripts, triggered by PostgreSQL functions to do the replication. Although it is written in Python, it does the job right and provides minor performance advantages over Slony.

Table 10 sums up the performance of each framework measured in a number of transactions per second:

Test performed	*Concurrent clients*	*PGPool*	*Slony*	*Londiste*
write	1	42	161	227
	5	80	305	320
	10	101	386	401
modify	1	210	655	671
	5	290	799	805
	10	311	933	1051
read/write	1	90	324	358
	5	137	570	593
	10	154	649	707

Table 10: Performance comparison - test results

Table 11 lists important clustering features supported by the tested frameworks:

Feature	*PGPool*	*Slony*	*Londiste*
Configuration and Installation	Easy	Easy to medium	Easy to medium
Community support	Very good	Very good	Good
Ease of maintenance	Very easy	Medium	Medium
Supported features	Rich	Medium to rich	Poor
Replication method used	Sync.	Async.	Async
Load balancer	Yes	No	Using SkyTools
Connection pooler	Yes	No	Using SkyTools
Query partitioner	Yes	No	No
Failover solution	Yes	No	No
Replication performance	Very slow	Fast	Fast
Schema replication	Yes	No	No
Replication sets support	No	Yes	Yes
Non-deterministic funcs	No	Yes	Yes

Table 11: PostgreSQL clustering approaches comparison

Although Slony and Londiste do not have a load balancer, connection pooler and do not provide any failover management in case a master node goes down, it is still possible to use a third party frameworks to provide this

functionality, e.g., one possible solution would be to use PGPool itself for this purpose and use Slony or Londiste for replication, as we demonstrated in our tests. SkyTools is a good alternative too, although we encountered some problems under Ubuntu during the installation.

It has to be underlined, that using one or another solution highly depends on a specific situation and user requirements for a cluster. The good thing about Open Source Software is, that a user could bundle and adapt the frameworks to his specific needs, thus building a custom solution which does the job right. It is not difficult, if needed, to design a cluster and use any of the reviewed frameworks, but it is rather important to know the limitations and drawbacks of a solution in advance.

6. Outlook and conclusion

The goal of this research paper was to contribute to the understanding of database replication technologies available on the market today. We outlined the theoretical background of database replication, provided the main definitions and identified possible problems. Performance, availability and management issues were spelled out and addressed. Two popular commercial DBMS and their OS counterparts were thoroughly reviewed, their features compared and the clustering approaches analyzed.

The research revealed, that the commercial offering from IBM and Oracle are much more advanced, feature-full and mature when it comes to a cluster implementation. These products provide complete clustering solutions which meet all the pre-defined criteria including load balancing, query partitioning, connection pooling and transparent failover management. The OS products, on the other hand, have their limitations: MySQL Cluster offers only asynchronous replication and PostgreSQL does not provide any clustering product, but rather relies on third party frameworks.

We investigated the available clustering frameworks for PostgreSQL (commercial and Open Source ones) and chose the most suitable ones based on the defined criteria – PGPool, Slony and Londiste. These frameworks were then stress tested, their performance analyzed and their advantaged and drawbacks outlined. As the frameworks in question use different replication methods, it was also possible to show the differences in performance and propose the areas of their usage. Synchronous replication, as provided by PGPool, is much slower than the asynchronous one, but it does guarantee data consistency at any time. On the other hand, it makes sense to use synchronous replication only if network availability could be guaranteed. Londiste and Slony provide asynchronous replication and deliver both acceptable performance. These solutions, however, provide neither a load balancer nor a connection pooler. Failover management is not included in these products as well. The PGPool framework could be setup as a higher layer for Londiste and Slony in order to compensate for the missing features,

but at the same time it cannot handle non-deterministic functions properly. Such a limitation forces a developer to adapt the database schema accordingly or to use SkyTools as an alternative. To draw the line, the research revealed the following weak points in tested frameworks:

- each framework supports only one replication method, i.e., synchronous or asynchronous.
- PGPool does not provide full non-deterministic functions support.
- Slony and Londiste do not replicate database schema changes automatically.
- load balancing, connection pooling and failover management must be configured separately in Slony and Londiste.

To conclude, we have to stress, that despite of the fact, that the chosen frameworks could not be directly seen as competitors to the commercial offerings from IBM and Oracle, it is still possible to implement a feature-full and reliable database cluster using the combination of these products. We also suggested a couple of approaches for implementing a PostgreSQL clustering solution based on SkyTools or PGPool for cluster management and Slony or Londiste for data replication. It must be noted, that despite of the fact, that the Postgres-R framework was not tested and compared in this paper, as it is still in development, the solution per-se looks very promising.

A. Appendix: PGPool configuration

The IP addresses of all the nodes are defined in the system */etc/hosts* file

```
172.16.201.132   vus2
172.16.201.133   vus3
```

Listing 2: /etc/hosts

The default pgpool installation has to be modified, as listed bellow:

```
# Replication mode
replication_mode = true

# Load balacing mode, i.e., all SELECTs except in a transaction block
# are load balanced. This is ignored if replication_mode is false.
load_balance_mode = true

# If true, operate in master/slave mode.
master_slave_mode = false

# IMPORTANT, especially using jdbc drivers
ignore_leading_white_space = true

# system DB info
system_db_hostname = 'localhost'
system_db_port = 5432

# database master node
backend_hostname0 = 'vus'
backend_port0 = 5432
backend_weight0 = 1

# database slave 1
backend_hostname0 = 'vus2'
```

```
backend_port0 = 5432
backend_weight0 = 1

# database slave 2
backend_hostname0 = 'vus3'
backend_port0 = 5432
backend_weight0 = 1
```

Listing 3: /etc/pgpool.cfg

B. Appendix: Slony configuration

The following script defines the cluster and subscription sets:

```
#!/bin/sh

CLUSTERNAME='cluster_replica'
RUSER='postgres'

DBNAME='replica'
MASTERHOST='vus'
SLAVE1='vus2'
SLAVE2='vus3'

# installing pl/pgSQL procedural language
createlang -U $RUSER -h vus plpgsql replica
createlang -U $RUSER -h vus2 plpgsql replica
createlang -U $RUSER -h vus3 plpgsql replica

slonik << _EOF_
#--
# Define cluster and nodes
#--
```

```
cluster name = $CLUSTERNAME;
node 1 admin conninfo = 'dbname=$DBNAME host=$MASTERHOST
    user=$RUSER';
node 2 admin conninfo = 'dbname=$DBNAME host=$SLAVE1 user=
    $RUSER';
node 3 admin conninfo = 'dbname=$DBNAME host=$SLAVE2 user=
    $RUSER';

#--
# Init master node
#--
init cluster ( id=1, comment = 'Master DB');

#--
# Define a replication set
#--
create set (id=1, origin=1, comment='Replication Set');
set add table (set id=1, origin=1, id=1, fully qualified name='public.
    client');
set add table (set id=1, origin=1, id=2, fully qualified name='public.
    item');
set add table (set id=1, origin=1, id=3, fully qualified name='public.
    client_item');

#--
# Create slave1
#--
store node (id=2, comment = 'Slave1');
store path (server = 1, client = 2, conninfo = 'dbname=$DBNAME
    host=$MASTERHOST user=$RUSER');
store path (server = 2, client = 1, conninfo = 'dbname=$DBNAME
    host=$SLAVE1 user=$RUSER');
```

```
#--
# Create slave2
#--
store node (id=3, comment = 'Slave2');
store path (server = 1, client = 3, conninfo = 'dbname=$DBNAME
    host=$MASTERHOST user=$RUSER');
store path (server = 3, client = 1, conninfo = 'dbname=$DBNAME
    host=$SLAVE1 user=$RUSER');
_EOF_

# Start replication engine
slon $CLUSTERNAME "dbname=$DBNAME user=$RUSER host=
    $MASTERHOST"
```

Listing 4: slony_setup.sh

After the daemons are started, the nodes have to subscribe for the set, as outlined before:

```
#!/bin/sh

CLUSTERNAME='cluster_replica2'
RUSER='postgres'

DBNAME='replica'
MASTERHOST='vus'
SLAVE1='vus2'
SLAVE2='vus3'

slonik <<_EOF_

cluster name = $CLUSTERNAME;

node 1 admin conninfo = 'dbname=$DBNAME host=$MASTERHOST
    user=$RUSER';
```

```
node 2 admin conninfo = 'dbname=$DBNAME host=$SLAVE1 user=
    $RUSER';
node 3 admin conninfo = 'dbname=$DBNAME host=$SLAVE2 user=
    $RUSER';

subscribe set ( id = 1, provider = 1, receiver = 2, forward = no);
subscribe set ( id = 1, provider = 1, receiver = 3, forward = no);
_EOF_
```

Listing 5: slony_subscribe.sh

After executing of this script, the slaves start the replication and apply all the changes continuosly.

C. Appendix: Londiste configuration

Ticker has to be configured and started in order to reference master database as follows:

```
[pgqadm]
job_name = ticker_replica
db = dbname=replica user=postgres

# how often to run maintenance [seconds]
maint_delay = 600

# how often to check for activity [seconds]
loop_delay = 0.1
 logfile = ~/londiste/log/%(job_name)s.log
 pidfile = ~/londiste/log/%(job_name)s.pid
```

Listing 6: ticker.ini

Each subscriber (slave node) needs a replication daemon and therefore a configuration file for londiste has to be defined:

```
[londiste]
job_name = replica_subcriber

provider_db = dbname=replica user=postgres port=5432 host=vus
subscriber_db = dbname=replica user=postgres port=5432 host=vus2

# it will be used as sql ident so no dots/spaces
pgq_queue_name = qreplica

logfile = ~/londiste/log/%(job_name)s.log
pidfile = ~/londiste/log/%(job_name)s.pid
```

Listing 7: master_to_slave.ini

References

Literature

[AAA+96] Alonso, Gustavo; Agrawal, Divyakant; Abbadi, Amr E.; Kamath, Mohan; Guenthoer, Roger; Mohan, C.: Advanced Transaction Models in Workflow Contexts. In: *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*. Washington, DC, USA : IEEE Computer Society, 1996, p. 574–581

[AAAS97] Agrawal, Divyakant; Alonso, Gustavo; Abbadi, Amr E.; Stanoi, Ioana: Exploiting Atomic Broadcast in Replicated Databases. In: *Euro-Par '97: Proceedings of the Third International Euro-Par Conference on Parallel Processing*. London, UK : Springer-Verlag, 1997, p. 491–507

[ABKW98] Anderson, Todd; Breitbart, Yuri; Korth, Henry F.; Wool, Avishai: Replication, consistency, and practicality: are these mutually exclusive? In: *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1998, p. 472–495

[AEAS97] Agrawal, D.; El Abbadi, A.; Steinke, R. C.: Epidemic algorithms in replicated databases. In: *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. New York, NY, USA : ACM, 1997, p. 161–175

[ANY04] Agrawal, Sanjay; Narasayya, Vivek; Yang, Beverly: Integrating vertical and horizontal partitioning into automated physical database design. In: *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2004, p. 359–370

[BBG+95] Berenson, Hal; Bernstein, Phil; Gray, Jim; Melton, Jim; O'Neil, Elizabeth; O'Neil, Patrick: A critique of ANSI SQL isolation levels. In: *SIGMOD '95: Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1995, p. 1–10

[BHG87] Bernstein, Philip A.; Hadzilacos, Vassco; Goodman, Nathan: *Concurrency control and recovery in database systems*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1987

[BK97] Breitbart, Yuri; Korth, Henry F.: Replication and consistency: being lazy helps sometimes. In: *PODS '97: Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. New York, NY, USA : ACM, 1997, p. 173–184

[BKR+99] Breitbart, Yuri; Komondoor, Raghavan; Rastogi, Rajeev; Seshadri, S.; Silberschatz, Avi: Update propagation protocols for replicated databates. In: *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1999, p. 90–108

[CRR96] Chundi, Parvathi; Rosenkrantz, Daniel J.; Ravi, S. S.: Deferred Updates and Data Placement in Distributed Databases. In: *ICDE '96: Proceedings of the Twelfth International Conference on Data Engineering*. Washington, DC, USA : IEEE Computer Society, 1996, p. 469–476

[EN03] Elmasri, Ramez; Navathe, Shamkant B.: *Fundamentals of Database Systems, Fourth Edition*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2003

[FAA99] Ferhatsomanoglu, Hakan; Agrawal, Divyakant; Abbadi, Amr E.: Clustering Declustered Data for Efficient Re-

trieval. In: *the Conference on Information and Knowledge Management*. Kansas City, Missouri, USA : ACM, November 1999, p. 343–350

[GBC+02] Gu, Lijun; Budd, Lloyd; Cayci, Aysegul; Hendricks, Colin; Purnell, Micks; Rigdon, Carol: *A practical guide to db2 udb data replication v8*. Riverton, NJ, USA : IBM Corp., 2002

[GG97] Gay, Jean-Yves; Gruenwald, Le: A Clustering Technique for Object-Oriented Databases. In: *DEXA '97: Proceedings of the 8th International Conference on Database and Expert Systems Applications*. London, UK : Springer-Verlag, September 1997, p. 81–90

[GGK+05] Gervasi, Osvaldo; Gavrilova, Marina L.; Kumar, Vipin; Lagana, Antonio; Lee, Heow P.; Mun, Youngsong; Taniar, David; Tan, Chih Jeng K.: *Computational Science and Its Applications - ICCSA 2005: International Conference, Singapore, May 9-12, 2005, Proceedings, Part I (Lecture Notes in Computer Science)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2005

[GHOS96] Gray, Jim; Helland, Pat; O'Neil, Patrick; Shasha, Dennis: The dangers of replication and a solution. In: *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 1996, p. 173–182

[HSAA99] Holliday, J.; Steinke, R.; Agrawal, D.; Abbadi, A. E. Epidemic Quorums for Managing Replicated Data. Santa Barbara, CA, USA. 1999

[HT93] Hadzilacos, Vassos; Toueg, Sam: *Fault-tolerant broadcasts and related problems*. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., 1993

[JP03] Jmaiel, Mohamed; Pepper, Peter: Development of communication protocols using algebraic and temporal specifications. In: *Comput. Netw.* 42 (2003), Nr. 6, p. 737–764. – ISSN 1389–1286

[KGK95] Ketterlin, A.; Gancarski, P.; Korczak, J.J.: Conceptual Clustering in Structured Databases: a Practical Approach. In: *In Proc. of the 1st Internationall Conf. On Knowledge Discovery and Data Mining.* Quebec, Montreal : AAAI, 1995, p. 21–185

[KKH08] Kline, Kevin; Kline, Daniel; Hunt, Brand: *SQL in a Nutshell.* O'Reilly Media, Inc., 2008

[Kyt05] Kyte, Thomas: *Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions.* Berkely, CA, USA : Apress, 2005

[Ler07] Lerner, Reuven M.: Open-source databases, Part III: choosing a database. In: *Linux J.* 2007 (2007), Nr. 158, p. 17. – ISSN 1075–3583

[Mat97] Mattison, Rob: *Understanding Database Management Systems.* New York, NY, USA : McGraw-Hill, Inc., 1997

[MZSMM03] Marculescu, Diana; Zamora, Nicholas H.; Stanley-Marbell, Phillip; Marculescu, Radu: Fault-Tolerant Techniques for Ambient Intelligent Distributed Systems. In: *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design.* Washington, DC, USA : IEEE Computer Society, 2003, p. 348

[Oja08] Oja, Asko. plProxy, pgBouncer, pgBalancer. Paritioning databases and using remote calls. Ottawa, Ontario, Canada. 2008

[PMS99] Pacitti, Esther; Minet, Pascale; Simon, Eric: Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. In: *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1999, p. 126–137

[PSM98] Pacitti, Esther; Simon, Eric; Melo, Rubens: Improving Data Freshness in Lazy Master Schemes. In: *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*. Washington, DC, USA : IEEE Computer Society, 1998, p. 160–171

[Rec01] Record, ACM S.: Using a Cluster Manager in a Spatial Database System. In: Aref, Walid G. (Hrsg.): *Proceedings of the ninth ACM international symposium on Advances in geographic information systems* Bd. 27. Atlanta, GA, USA : Brinkhoff, 2001, p. 74–153

[SAA98] Stanoi, I.; Agrawal, D.; Abbadi, A. E.: Using Broadcast Primitives in Replicated Databases. In: *ICDCS '98: Proceedings of the The 18th International Conference on Distributed Computing Systems*. Washington, DC, USA : IEEE Computer Society, 1998, p. 145–155

[Tes07] Test, T. Jeffrey: *Surviving the next Database Disaster*. Saarbruecken, Germany, Germany : VDM Verlag, 2007

[Tum04] Tumma, Madhu: *Oracle Streams: High Speed Replication and Data Sharing (Oracle In-Focus Series)*. Rampant TechPress, 2004

[Wes07] Weske, Mathias: *Business Process Management: Concepts, Languages, Architectures*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2007

[ZSG04] Zaman, Mujiba; Surabattula, Jyotsna; Gruenwald, Le: An Auto-Indexing Technique for Databases Based on Clustering. In: *DEXA '04: Proceedings of the Database and Expert Systems Applications, 15th International Workshop*. Washington, DC, USA : IEEE Computer Society, 2004, p. 776–780

Web sites

[Buc] Bucardo: *Bucardo, asynchronous master-master and master-slave replication system*. Bucardo, http://bucardo.org/bucardo.html, last checked: 04.06.2009

[Com] Command Prompt, Inc.: *Mammoth PostgreSQL replication system*. Command Prompt, Inc., http://www.commandprompt.com/products/mammothreplicator/, last checked: 04.06.2009

[Cyb] Cybercluster: *CyberTec, Cybercluster for PostgreSQL*. Cybercluster, http://www.postgresql.at/english/pr_cybercluster_e.html, last checked: 02.06.2009

[IBMa] IBM: *Lock avoidance in DB2 UDB V8*. IBM, http://www.ibm.com/developerworks/data/library/techarticle/dm-0509schuetz/, last checked: 15.09.2009

[IBMb] IBM Corporation: *IBM DB2 Information Center*. IBM Corporation, http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp, last checked: 15.04.2009

[Oraa] Oracle: *Oracle Real Application Cluster, White Paper*. Oracle, http://www.oracle.com/technology/products/database/clustering/pdf/twp_rac11g.pdf, last checked: 21.05.2009

[Orab] Oracle: *Oracle Streams Replication, An Oracle White Paper*. Oracle, http://www.oracle.com/technology/products/dataint/pdf/twp_streams_replication_11gr1.pdf, last checked: 21.06.2009

[PGC] PGCluster: *PGCluster, The Multi-master and synchronous replication system.* PGCluster, http://pgcluster.projects.postgresql.org, last checked: 17.05.2009

[Posa] Postgres-R Foundation: *Terms and Definitions for Database Replication.* Postgres-R Foundation, http://www.postgres-r.org/documentation/terms, last checked: 13.05.2009

[Posb] PostgreSQL Foundation: *PGPool Introduction for beginners.* PostgreSQL Foundation, http://pgpool.projects.postgresql.org, last checked: 05.06.2009

[Posc] PostgreSQL Global Development Group: *PostgreSQL Online Documentation.* PostgreSQL Global Development Group, http://www.postgresql.org/docs/8.3/static/index.html, last checked: 27.05.2009

[Skya] Skype Developer Zone: *Lightweight connection pooler for PostgreSQL.* Skype Developer Zone, https://developer.skype.com/SkypeGarage/DbProjects/PgBouncer, last checked: 20.05.2009

[Skyb] Skype Developer Zone: *SkyTools.* Skype Developer Zone, https://developer.skype.com/SkypeGarage/DbProjects/SkyTools, last checked: 22.05.2009

[Slo] Slony-I: *Slony-I, enterprise-level replication system.* Slony-I, http://www.slony.info/, last checked: 26.05.2009

[Sun] Sun Microsystems, Inc: *MySQL Online Documentation.* Sun Microsystems, Inc, http://dev.mysql.com/doc/#cluster, last checked: 03.06.2009

[Ver] Versant: *Database scalability and clustering.* Versant, http://www.versant.com/developer/resources/objectdatabase/whitepapers/vsnt_whitepaper_scalability_clustering.pdf, last checked: 13.08.2009

[Wik] WIKIPEDIA: *Oracle Clusterware*. Wikipedia, http://en.wikipedia.org/wiki/Oracle_Clusterware, last checked: 16.08.2009