# An Adaptive Replication Environment for a Relational Database Management System

Bruno Cesar Neves de Oliveira[1]
Instituto Federal da Bahia
Salvador, Bahia – Brazil
Email: brunocn.oliveira@gmail.com

***Abstract*—Database replication has been a constantly discussed and studied topic. Replication is a fundamental factor that comply the companies' needs that leads with relevant information. Applying database replication in a right way with well-defined interfaces guarantees that high availability, performance and connectivity are maximized. When we design a replication database environment, group communication and fault tolerance are important aspects that must be considered so that DBMS features are sustained, such as consistency, integrity, security and others. This paper presents an approach to apply and adaptable replication in a DBMS. This means that a database should support different replication strategies and allow, at runtime, that the strategy adopted could be changed. For each scenario or specific system situation (either an enterprise environment or a shared cloud), replica nodes provide an interface that enables to adjust the way that replication is done – adapting to the needs of certain instant.**

***Keywords*—Database, fault tolerance, group communication, replication.**

***Resumo—Replicação em Sistemas de Gerenciamento de Banco de Dados (SGBD) tem sido um tópico cada vez mais discutido e estudado. A replicação é o fator primordial para atender as necessidades das empresas que lidam com informações relevantes. Aplicar a replicação em banco de dados de forma correta com interfaces claras e bem definidas garante que a alta disponibilidade, desempenho e conectividade sejam maximizados. Projetar um ambiente de banco de dados replicável requer que aspectos como comunicação em grupo e tolerância a falhas sejam considerados de modo que as características do SGBD sejam mantidas, tais como consistência, integridade, segurança, dentre outros. Este artigo apresenta uma abordagem para aplicar replicação adaptável em um SGBD. Isto significa que um banco de dados deve suportar diferentes estratégias de replicação e permitir, em tempo de execução, que a estratégia adotada seja alterada. Para cada cenário ou situação específica do sistema (seja em um ambiente corporativo ou em uma nuvem compartilhada), os nós das réplicas fornecem uma interface que possibilita ajustar a forma como é feita a replicação, se adaptando às necessidades momentâneas.***

***Palavras-chave*—Banco de dados, tolerância a falhas, comunicação em grupo, replicação.**

[1]This work is part of the requirements for certification as Specialist on Ubiquitous and Distributed Computing under supervision of professor Allan Edgard Silva Freitas at Instituto Federal da Bahia.

## CONTENTS

## I. INTRODUCTION

AN increasingly valuable element to people and organizations around the world is information. Modern society has been living over information era, where data must be constantly present, handled and processed as fast as possible [1]. Information is becoming so important today, mainly in business, that a slight failure in data access may cause a huge problem. For instance, if a telecommunication platform enters in bypass mode (because the environment is unavailable) or a bank system goes down, the company will most likely suffer a considerable loss of revenue due to data unavailability.

If a database that stores all data of a company goes down, critical information can be lost or become inaccessible so it cause significant damage. In order to deal with data unavailability and its consequence and also to overcome the potential problems caused by centralized DBMS (Database Management System) approach, we use data replication approach. Such approach is discussed in Section II.

Database replication has been a frequent feature in DBMS scope for the last years. In this context, each database is autonomous, allowing tuning for the best fit configuration to a specific scenario, as well as increasing the performance of each individual DBMS replica according to their needs.

Furthermore, the modularity is another important aspect that must be considered. Using database replication, it is possible to create well-defined interfaces that connect several flavors of database and can, moreover, make use of different types of replication protocols. To reach these goals, it is necessary to provide: (a) a "friendly" well-defined replication interfaces; (b) group communication support; and (c) "pluggable" (or connectable) replication protocol. In other words, they can be replaced in the architecture seamlessly without large impacts [2].

It is possible to achieve a suitable architecture for DBMS replication joining these three requirements. Well-defined interfaces and "pluggable" protocols, allows attains the modularity and flexibility. On the other hand, it is possible to obtain reliable delivery of messages to all system replicas through the adoption of a mature group communication protocol.

Fault tolerance and the group communication techniques are intimately linked and give the foundations to replication. Database replication has been constantly debated as an efficient way to increase high availability and performance on data retrieval. Given a replicable database environment it is possible that each DBMS has its own configurations and tunings, though information is always consistent throughout the system nodes.

The advent of cloud computing [3] and its fast growth is transforming the information technology industry. Many companies are considering convenient to host their DBMS to the cloud letting to it the duty of data management, including all required resources to accomplish the desired performance with high availability. It is also important that each replica node interact harmonically with the cloud or even more: provide effortless mechanisms to reach the best performance.

An extremely useful mechanism to the cloud might be the support of different replication strategies in the same database. Aligned with this, allows changing these replication strategies dynamically (at runtime) with no additional builds (compilations), makes the environment adaptable and save costs to the cloud hosting and customers.

Therefore, it is essential to study and develop new techniques driven to the practice of replication in databases in order to achieve the desired high availability and performance of distributed enterprise systems.

### A. *Objectives*

The purpose of this project is to develop a general middleware that can be attached to a DBMS in order to replicate database operations through different replicas located in several sites. This middleware is based on group communication protocol to allow total order delivery [4] of SQL instructions to all replica nodes.

Additionally, two replication strategies are implemented into the same database in order to make a comparison of both methods immersed in the same environment. The main idea is to have a database able to support both active and passive replication techniques. This becomes an advantage, since we can adapt replication strategy to be used according to a desired service level, once that most of the DBMS support just one strategy for replication.

Finally, a performance evaluation of both replication implementations is done. The contribution of this paper is therefore the valuable capability of switching replication strategy at execution time. Thus, depending on the environment behavior, the database can behave differently in order to enhance performance and save so much effort and resources.

### B. *Paper structure*

The rest of this paper is organized as follows. Section II gives a background discussion concerned on the main topics related. This section identifies the reasons of using distributed and replicable databases, distinguishing them from centralized method. It also presents how replication is related to distributed computing fundamentals, addressing to fault tolerance and group communication – mighty important and applied on replication strategies.

Section III clarifies the essential notions of database replication strategies to understand how they work. It is noteworthy to point out these concepts so as it gives foundation in understanding further the implementation and benchmarking comparison of both active and passive methods.

Section IV presents the development infrastructure. It shows the DBMS used throughout the project (H2DB) and the group communication toolkit adopted (JGroups).

The implementation details as well as each part of the generic architecture for database replication are described in Section V.

Section VI evaluates the replication environment with a benchmarking approach widely used: TPC-C. It also discusses a comparison of both replication strategies and the centralized mode (database with no replication) and shows the results drawn during the tests.

Finally, Section VII summarizes the entire project and draws a conclusion reached on this paper as well as a discussion of future directions.

## II. BACKGROUND

This section surveys the most important concepts of database environment to contextualize replication database as well as the main related subjects in this area, such as fault tolerance and group communication.

### A. *Database environment: An overview of centralized and distributed DBMS*

The primary and necessary tool for the storage of relevant information in a corporation is a database. Regarding relational database it is important to ensure the data consistency. Thus, there are some concerns such as independence, control and data integration, treatment of redundant information, and others, which are featured in a database management system. All these features can be wrapped into a single centralized DBMS, as well as in many geographically distributed servers.

Since database has several important features, it is appropriate to host the DBMS on a dedicated server and let it always available to users, as well as all external systems that access the database. The DBMS presents significant advantages of security primitives and severe restrictions to their access. There are some drawbacks when it is used in a single centralized environment, though. In this case, if there is an interruption in the host server, or even a failure in the software itself, the DBMS will be unavailable. Thus, all users and systems that need database will be, inexorably, blocked from accessing any information - until the failure be remedied and the database back to normal operation.

On the other hand, the idea of distributing data geographically appears in objection to leave them in a single centralized server. In this direction, all data will be scattered throughout the distributed network, each one with their own storage devices. Thus, the storage devices are not attached to a common processing unit such a CPU and can exchange messages through networked computers [5].

It should also have transparency in information access, that is, it is not required from users – when they perform a query or manipulate data – a prior knowledge of where all the necessary data are specifically stored. The customer just performs a query and data is returned, regardless of whether the query was requested on the machine/server where the data was persisted. It might be extremely complex to make possible this distribution and transparency location of information. To perform this role easier, there is already an approach called Distributed Database Management System, or simply Distributed DBMS. Elmasri and Navathe [6] define a distributed database as a collection of multiple databases logically interrelated and distributed across a computer network.

In spite of the fact that it presents more complexity and requires a higher cost of implementation, distributed DBMS ensures fairly data availability and modularity, when compared with centralized DBMS. Even though distributed DBMS brings up a good alternative to keep the data more available for users than centralized databases, it still has some shortcomings.

When one or more servers fails, some information, which was stored in one of these faulty servers, may be inaccessible. Additionally, the availability – in some cases – can be affected when, for example, a query is performed with significant joins of geographically distant data. For instance, if the query contains three joins within three tables (hosted in different places), it will have a performance decrease owing to the huge flow of sending messages to change information between the servers.

As seen, distributed DBMS still has some unsatisfactory flaws when high availability and performance are required indeed. For these reasons, it is necessary – in robust systems that require a high availability of data – the usage of multiple identical DBMS that communicate each other harmonically in order to maintain the consistency and integrity without loss of availability. This approach is known as replicated database and is described next.

## *B. Replicated Databases*

One of the approaches adopted to increase the availability of a system/database keeping natural attributes (such as consistency, integrity, security, etc.) is named replication. The aim of replication is to provide the combinations of high availability, high integrity, scalability and, in some cases, enhance the performance of both systems and database. This is achieved due to the creation of multiple copies of a possibly mutating data object, such as a file, a database, or other kind of data source [8]. Replication therefore provides effective ways to increase availability and improve fault tolerance in distributed systems.

In order to accomplish a replication environment in a set of database there must be redundancy of information among all nodes (replicas) within the environment. According to Junior-Alfranio [7], "*redundancy is a key element to provide fault-tolerant applications with increased performance*". Namely, it is possible to enhance performance and avoid failures on robust systems using redundancy.

The strategy adopted to take place redundancy in practice is replication. Replication can be implemented in database systems by two or more DBMSs that provide the same schema and data and are able to communicate each other through the network. Thus, replication is considered as a method to perform data duplication.

In a replication database environment, several replicas are needed so that high availability and integrity can be achieved. Also, the use of several replicas enables to deal with a number of independent failures. Hence, each DBMS may have their particular configuration of hardware and software, even though all of them store the same data item.

As stated by Renesse [8], in order to achieve high performance it is necessary to use a sufficient number of replicas so that it is possible to meet the load imposed on replicated objects. That is, depending on the business needs, the more data availability desired, more replicas can be included within the environment.

There are two models of database replication that might be applied in a production environment: 1) primary backup and 2) update anywhere. Each of these two models holds their own advantages and challenges depending on the environment where they can be applied [9].

The first requires a master node for the entire replication environment. This master node is responsible for receiving requests from any sources and replicate transactions to other nodes in the network. The remaining replicas of the environment, therefore, are read-only. When a node needs an update, it requests the master to perform the update process. All updates emanate from a master copy of the object. As shown in

Figure 1 (a), clients communicate with the master copy and this one takes care of updating the others.

On the other hand, the update anywhere model enables that any node in the environment process a request from the client and update a data item. The other copies are updated by the node which received that particular request with the update transaction. Because there is no master, replicas can communicate with any node in the environment. In this case, updates may emanate from anywhere, namely, clients can communicate with any node. This model is illustrated in Figure 1 (b).

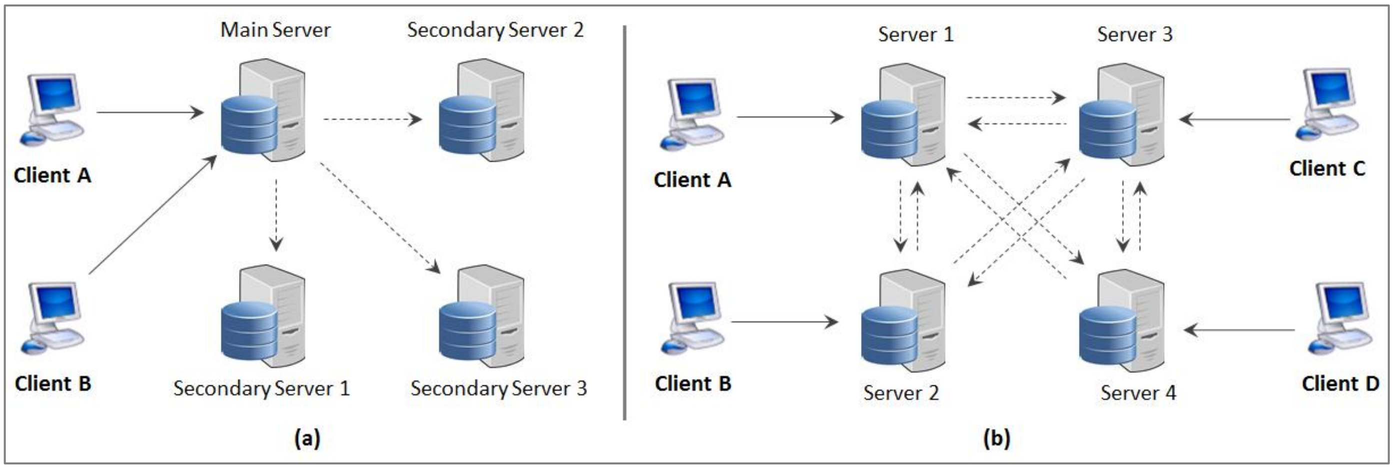


Figure 1.Models of Replication: a) Primary-backup;  b) Update anywhere

As a result, the replication database has some advantages over a purely distributed database with no replication. Some of these advantages are listed as follows:

- Increasing data availability, as any data can be accessed in different places;
- Increasing reliability, since data can be accessed even when there is unavailability of one node;
- Reducing network traffic at peak times, since replication can be scheduled to happen at specific times with less traffic of data;
- Improving response time for search and data aggregation;
- Transparency in redundancy of information and to clients' accesses.

Some relational database management systems (RDBMS) offered by renowned companies have already native replication support, for example, Oracle Replication [10] and SQL Server Replication [11]. Nevertheless, the replication model is strictly closed and hard to change and adapt to business needs. The customers and developers become hostage to its closed technology and protocols used by information sharing and group communication. On the other hand, when developers build their own replication methods, they gain freedom and autonomy to optimize their resources as well as approach it from the customer's needs (such as performance and security) in a convenient way.

It is possible to implement decentralized applications, optimize performance through load balance or even direct the request to be processed into a geographically closest server from the client.

### C. *Fault Tolerance*

To achieve availability and prevent disasters arising from the loss of important information or valuable data, it is necessary to apply mechanisms to tolerate system failures. These failures may have several causes and could be managed by software fault tolerance or hardware fault tolerance. While the software fault tolerance looks after software failures, hardware fault tolerance usually comes from hardware issues and is mostly discussed by the electrical engineering community [12]. Availability and reliability of information are both related and are included as part of the strategy for fault tolerance. The fault tolerance mentioned in this paper concerns on crash failures, since the most proposed techniques of replicable environments leads with this failure type.

The basic goal of fault tolerance is to fulfill the dependability aspects [12]. This seeks to enhance the safety and quality of service offered by the system, increasing its reliability, enlarging its availability and facilitating maintainability (among others, as shown in Figure 2 [12]). In other words, the system should behave as desirable avoiding potential failures and, additionally, increase the time which it is available to provide services to clients (either users or other systems).

In order to avoid absence or loss of information, it is required that the system stay always available. Availability is one of the attributes of dependability, as illustrated in Figure 2, and it is not cheap to achieve a high availability.

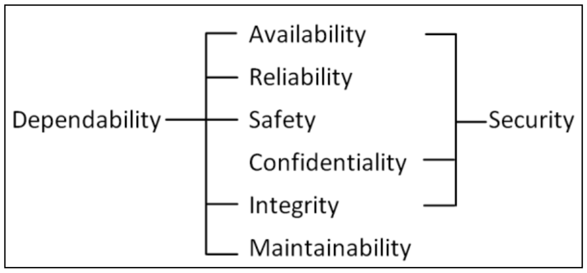


Figure 2. Dependability and security attributes (A. Avizienis, et al, 2004 [12], p.14)

There are many different terminologies found in the literature to refer the high availability [13]. A system can be featured as high available if it was designed and built with a sufficient number of components of hardware and software to assure its functionality. Furthermore, it must have enough redundancy in the components to prevent predetermined failures.

To implement and maintain redundancy of information, replication methods are used so that part of dependability (availability) can be achieved.

Dependability supports availability and this last one contains redundancy. Furthermore replication is used to duplicate the data, maintain redundancy and achieve the highest level of availability and dependability.

Redundancy strategy is used in traditional high availability platforms. If a system requires high availability and, consequently, it implies on redundant information, the replication of data becomes mandatory. To do so it is recommended to handle a set of databases which have duplicate data and apply the replication mechanism. Thus, this set of databases will form distributed and replicated databases which provide high availability. If one database replica fails in the distributed system, the other replicas may be able to operate and offer the information, since they have the same data as the first one that failed. Therefore, the failure of a site does not necessarily imply the shutdown of the entire system.

### *D. Group Communication*

In the context of distributed systems, it is important to have an effective communication among the involved processes. For communication occur, there must be a communication channel (typically the network) that enables computers (processes) exchange messages with each other. There is an abstraction to perform this communication mechanism as known as group communication (GC). Group communication is used by most distributed applications, especially for the replicated database systems, since the replicas (considered processes or nodes) are constantly changing messages such as transactions and data updating. The GC might be considered as a middleware between the layer which implements replication and the transport layer, as shown in Figure 3 [14].

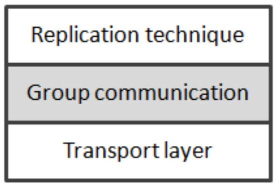


Figure 3. Group communication [14]

Group communication abstraction aims to solve basic problems of inconsistencies in communication among distributed processes that cooperates to perform the execution of some task. Thus, a group is basically a set of processes that cooperate each other to achieve a common goal. A group is also identified by a name so that messages can be sent to all members referencing the group identifier. Thus, the abstraction of GC is aware of the group members and delivers the message to all active processes (members) [14].

The communication among processes forming a group is accomplished through a mechanism called diffusion - a message to be transmitted is sent to all members belonging to the group. An important property of the diffusion process performed by group communication is atomicity. In general, group communication protocols ensure that a message, once delivered to a particular group process, is also delivered to all other processes running in the same group (atomicity), even if the process that originated the message fails before finalizing the transmission.

Another property, generally secured by GC abstraction, is the total order in which messages are delivered to different processes (total order). A total order protocol delivers messages in the same order for all processes in a group. For instance, the GC protocols aid in synchronize all the transactions and execute them inside each replica in the right sequence. This property has a profound importance for database replication environment, as it is required the isolation of each transaction performed by DBMS.

Using multicast communication, messages can be sent exactly to the group of machines that are interested in receiving the message. Thus, apart from getting ease the implementation of other abstractions, group communication service has also been held as a basic tool to programme general distributed applications. It eases the implementation of replication protocols by providing abstractions for message reliability, ordering and failure detection [2].

There are some different implemented abstractions for group communication provided by developers' community. The toolkit used in this project is described in section IV.

## III. DATABASE REPLICATION TECHNIQUES

Prior sections discuss the key points concerned to database replication environment and how they are related. Developers can create a replicable database environment applying the concepts of redundancy and fault tolerance working together with group communication techniques.

When data is replicated, atomicity and isolation need to be guaranteed. For atomicity there is no much problem, since it can be guaranteed by using 2 Phase Commit [15]. The challenge, however, is to ensure that serialization orders are the same at all sites. In other words, it is primordial to guarantee that all nodes execute the same operations exactly in the same order; otherwise the copies would be inconsistent.

Some techniques have been proposed in managing replicated data. An efficient and effective replication technique is decisive to improve both the availability and performance. Thus, data and transactions can be replicated aimed at failures recovery.

Replication can be classified according to the means that a set of replicas receives and processes requests from a client [16]. This section analysis the most common techniques used to replicate databases and compare each other: active replication versus passive replication. It gives therefore the understanding of implemented replication (discussed in next sections) and the comparative tests performed in both strategies that can be switched at runtime.

### *A. Active Replication*

The active replication, found sometimes in the literature as state-machine approach [7] [16], is a heavy replication mechanism which receives the requests from a client and send to all replicas to process in the same deterministic way, namely, all replicas updates the instructions in the same order. Then, the replica that received the client request sends back the response to the client.

This technique allows any node to update any local data, so it is based on Update-Everywhere replication model – as shown in Figure 1 (b). The front end (which could be one node that receives the request) sends the requests as a multicast message to the group of replication manager. All replication managers (hosted in each node) process the request independently, though it is done in an identical way. Supposing that a replica manager fails by crash, the service still works since the other replica managers can process and reply the requests naturally [16].

In this technique, a read operation is allowed to read any copy of data. Meanwhile, a write operation is required to write all copies of data. Even though this seems to be a great and elegant technique, it has a significant drawback that affects the whole system due to the high resource usage: performance. In this case, the replica managers demand high consumption of resources, since all nodes need to execute the same transaction before send the response to the client. It reduces update performance and increases the transactions response time [16].

On the other hand, according to [16], the main advantage of active replication is its simplicity (e.g., same code everywhere) and the failure transparency in view of the fact that they are fully hidden from the clients. Besides that, active replication keeps all sites exactly synchronized by updating all the nodes as part of one atomic transaction.

### B. *Passive Replication*

One of the most popular replication techniques is passive replication, also called Primary-backup technique. In this technique one of the replicas is designated primary. Generally, the designation of the primary replica is accomplished according to the node that has not crashed and that has the lowest identifier. The remaining nodes are called backups.

The front end (which could be one node that receives the request) communicates only with the primary replica manager. This executes the operations and sends the resulting state updates to each of the replicas (including itself), which, passively, apply the state updates in the order received. Thus, this technique is based on Primary-backup model shown in Figure 1 (a). The primary replica receives the requests from a client, processes them, and replies back to the client. Changes gathered in the execution are propagated to other replicas (backup) either in a lazy or eager approach [7].

In passive replication it is not necessary that operations be deterministic – the main disadvantage of active replication. Typically, the primary will resolve non-determinism and produce state updates, which are deterministic. If the primary fails, the client determines one backup to be promoted as the new primary to whom it retransmits its update.

Figure 4 shows the different ways that passive and active replication works on data propagation. Passive replication asynchronously propagates replica updates to other nodes after the updating transaction commits. Because it is faster, some systems that need to improve response time use passive replication, instead of active replication.

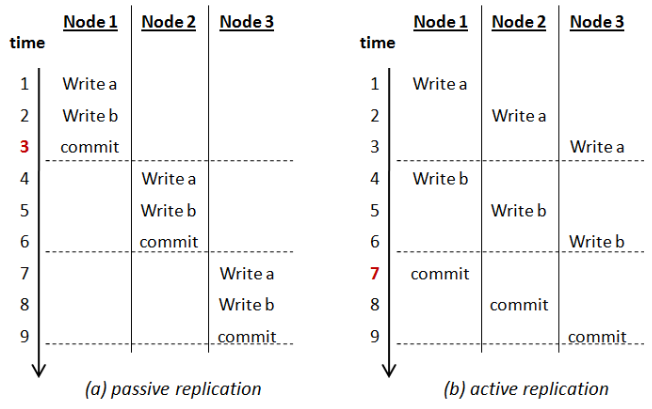


Figure 4. Data propagation: a) Passive replication; b) active replication

The example of Figure 4 shows an environment composed by 3 replica nodes that receives 3 operations: 1. Write a; 2. Write b; and 3. Commit. In passive replication, the Node 1 updates the whole transaction and then sends the operation to other nodes. Active replication updates all nodes for each operation, thus the replicas are always updated.

In contrast to the active replication, in passive replication there is no transparency to the clients when failures happen. It is necessary to guarantee that updates sent by the new and the faulty primary are received and applied in the same order in all replicas.

## IV. Development Infrastructure

This section discusses development infrastructure used to implement the database replicable environment. First it presents the database used and shows the reasons of why this has been chosen. Secondly, the group communication toolkit is presented.

### A. *DBMS – H2Database*

The simulation was implemented in H2 database which is a relational database management system written in Java and distributed under an open source license. It is a lightweight database which is shipped with JBoss AS (JBoss Application Server 7) [17] distribution and other reasonable important projects such as nWire[1]. It is also already supported by object-relational mapping (ORM) tools, for instance Apache OpenJPA and Hibernate ORM.

Because it is an open source lightweight database and it is written in a high level object oriented language (Java), H2DB becomes a convenient choice for developers to build and test applications with more efficiency and advanced configuration.

Figure 5 was drawn based on some benchmarks accomplished by H2 team [18].

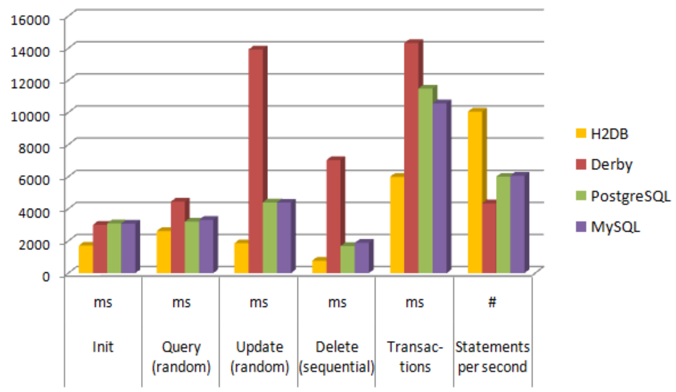


Figure 5. Sample client-server performance comparison of H2DB to other database engines. Reprinted from [18]

As it can be seen in Figure 5 H2 database overtops performance in many cases when compared to other similar open source databases. Besides its speed, there are other meaningful reasons to use H2DB.There are two different ways that H2DB can be used in, as follows:

1. Running in a server as a client-server mode (traditional way); and

2. Embedded in Java applications where data will not be persisted on the disk, just in memory. This mode is generally used in games or huge frameworks like JBoss AS that comes with a H2DB configured as in-memory database.

Thus, if it is necessary to get the best from H2 database it is appropriated to use a server mode database – which in fact exposes advanced features, for instance, exposes TCP/IP socket for other processes.

On the other hand, the embedded mode is practical and gives to developer the flexibility to install the software in a portable device or even share the database on a cloud environment.

As it is an open source project, it is also supported by open source community. H2DB is lightning fast for small to midsized databases so as it is suitable for the proposed project of replication. In addition, traditional open source databases such as MySQL and PostgreSQL also provide replication mechanisms already implemented.

Despite all these advantages, H2DB has its undesirable features as follows:

i. Code maturity: compared to the large databases such as Oracle, IBM DB 2, MS SQL Server, MySQL, PostgreSQL, the Java databases are relatively new and therefore not so stable.

ii. Commercial support: even though H2 database has a commercial support, it is not so wide and easy if compared to more renowned database.

In spite of that, H2DB remains advantageous to implement the replicable environment and comparative tests. Even though it is not recommended for large companies, it can be attractive useful for both development and test due to its flexible configuration. Table 1 presents a comparison among H2DB, Derby, MySQL and PostgreSQL database. Indeed, Table 1 points out some differences in features of these DBMSs.

| *Feature* | *H2DB* | *Derby* | *MySQL* | *PostgreSQL* |
|---|---|---|---|---|
| Embedded Mode | Yes | Yes | No | No |
| In-Memory Mode | Yes | Yes | No | No |
| Encrypted Database | Yes | Yes | No | No |
| ODBC Driver | Yes | No | Yes | Yes |
| Sequences | Yes | Yes | No | Yes |

[1]nWire – Software Visualization Tool. Available at: http://www.nwiresoftware.com

| | | | | |
|---|---|---|---|---|
| CLOB/BLOB Compression | Yes | No | No | Yes |
| Pure Java | Yes | Yes | No | No |
| Footprint (jar/dll size) | ~1.5 MB | ~3.0 MB | ~4.0 MB | ~6.0 MB |

Table 1. Comparison of H2DB to other database engines. Addapted from [18]

As illustrated in Table 1, H2DB competes tightly with other well-known concurrent DBMSs and takes advantages in many cases. For instance, H2 database supports encryption, affords an additional feature to run just in-memory and can be embedded within applications. These features, however, are not supported by MySQL and PostgreSQL. Additionally, Derby database lacks an ODBC Driver and does not have support for large objects such as BLOB and CLOB – features supported in H2 database.

### B. *Group communication – JGroups*

Group communication mechanism aids the development of distributed systems over a network. As we need to implement a replicable environment in a database system connected by network, we use the JGroups [19] toolkit for reliable messaging. It is necessary to implement replicas (known as nodes) which must be capable to exchange messages so that they can be always updated. JGroups is based on IP multicast and can aid us to create groups whose nodes can send messages to each other.

This project was implemented under Java platform so that we can take advantage of JGroups, since it is also written entirely in Java. Its main features include [19]:

- ✓ Group creation and deletion: it is possible to create a group with an identifier that contains all replica instances.
- ✓ Joining and leaving of nodes in the group: replicas can be added into groups at runtime, with minimum effort.
- ✓ Membership detection and notification about joined/left/crashed group nodes: when one replica crashes, it may rejoin the group and the others can send the transactions executed during this absent period.
- ✓ Detection and removal of crashed nodes.
- ✓ Sending and receiving of node-to-group reliable messages (point-to-multipoint): one replica can send its data item to the group so all the others replicas can update it.
- ✓ Sending and receiving of node-to-node reliable messages (point-to-point): one replica can send the updates to another past crashed replica that rejoined the group. In this case, the recovery process of one replica is performed.

As seen, JGroups is considered an important component of the environment that implements all the communications features through the networked computers. Therefore it fits perfectly with our needs to assure the reliable messaging among replica instances.

## V. Simulation-Based Approach

This section shows the architecture model to implement database replication. Further it is described the essential elements of the replication architecture and how they are implemented and coupled inside H2 database.

### A. *Architecture Model for Implementation*

The implementation was based on the replication architecture model for database suggested in [2]. Therefore, it is important to realize the main concepts of this architecture to understand the implementation.

This architecture model is essentially composed of seven elements, though three of them are indeed the core of replication. Figure 6 illustrates the model for replication architecture:

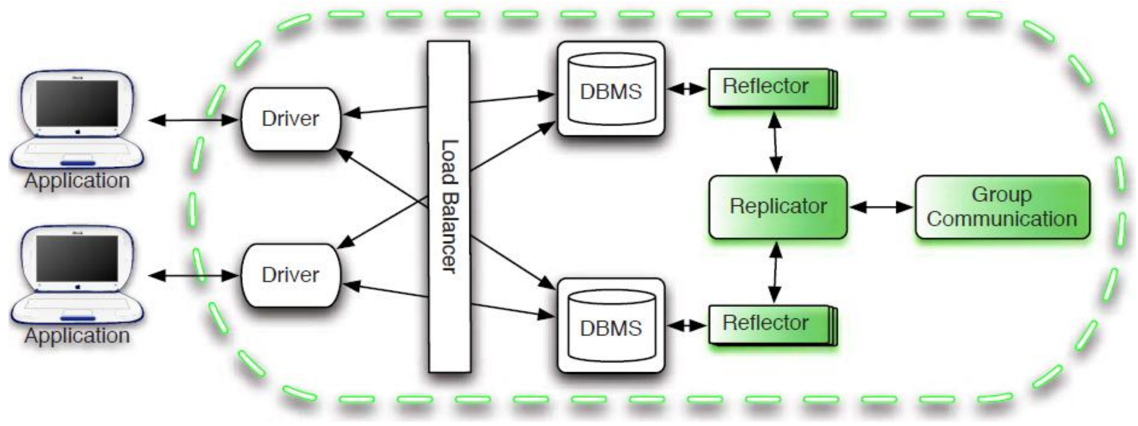


Figure 6. Replication architecture model. Adapted from [2]

The architecture shown in Figure 6 is based on [2] and consists of the following blocks:

- **The Application**, which might be the client tier and sends requests to database.
- **The Driver** which affords a standard interface for the Application tier. The Driver should provide remote accesses to the database using a low-level mechanism that ease the communication between client (application) and server (database).
- **The Load Balancer** dispatches client requests to database replicas using a suitable load-balancer algorithm. Although it is an important component, the Load Balancer is needless to apply the replication, namely, we can implement replication without the Load Balancer. However, it improves the system performance.
- **The DBMS**, which holds the database content and handles remote requests to query and modify data expressed in standard SQL.
- **The Reflector** is attached to each DBMS and allows inspection and modification of on-going transaction processing.
- **The Replicator** mediates the coordination among multiple reflectors in order to enforce the desired consistency criteria on the replicated database. This component uses the group communication mechanism to exchange messages among replicas.
- **The Group Communication** supports the communication and coordination of local Replicators.

The last 3 blocks are the key components which works together to achieve the replication. Thus, this project focuses specifically on these three components in order to implement a basic database replication environment.

Essential components of the architecture are the interfaces among the building blocks, which allow them to be reused in different contexts or easily switched to other implementation. This minimizes the coupling and enhances the system cohesion as well as its modularity.

In order to reach modularity without losing performance, it is imperative that the system has replication support from the database engine. The client interfaces provided by a DBMS do not afford enough information for replication protocols. The replication protocols must know details about the engine steps to perform a transaction in order to achieve good performance [2]. The interfaces exposed by the Reflector and Replicators as well as their implementation are detailed next.

### *B. Components Implementation: Initialization, Recovery and Replication*

As exposed previously, H2DB is a suitable database for this project. The architecture relies on such well-defined interfaces that interact with database implementation and group communication protocol using JGroups [19]. This section discusses the main concepts of implementation, such as initialization process, recovery and replication components. Figure 7 shows the functionality of the replicated DBMS initialization.

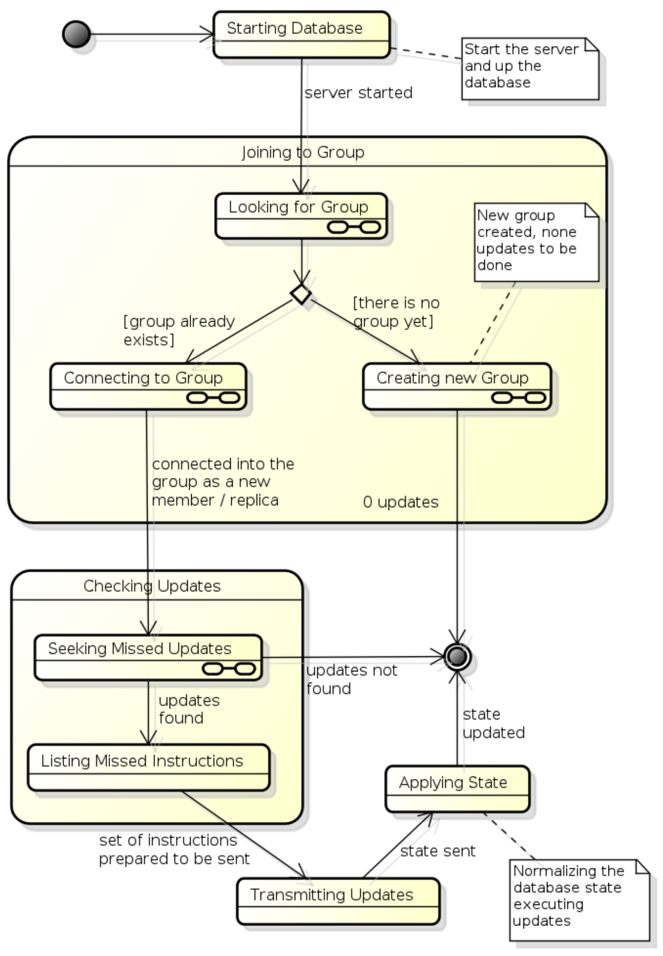


Figure 7. State machine diagram of DBMS initialization

Once the database initiates, a service listens to database requests. H2DB goes to “Joining to Group” state (Figure 7), that looks up to the JGroups replica group. If there is any such group, it will be created and the replica becomes its leader (master). Otherwise, if there is such group, database becomes a member of it as a new replica node within the environment.

After joined the group, database triggers to “Cheking Updates” state. In that state, it requests to leader updates from its current state: the leader receives the current state of a replica, checks if there is any updates to do and send a list of commands to new replica update its state in order to accomplish with leader state. This is part of the recovery synchronization mechanism and state replica is gathered from a transaction log.

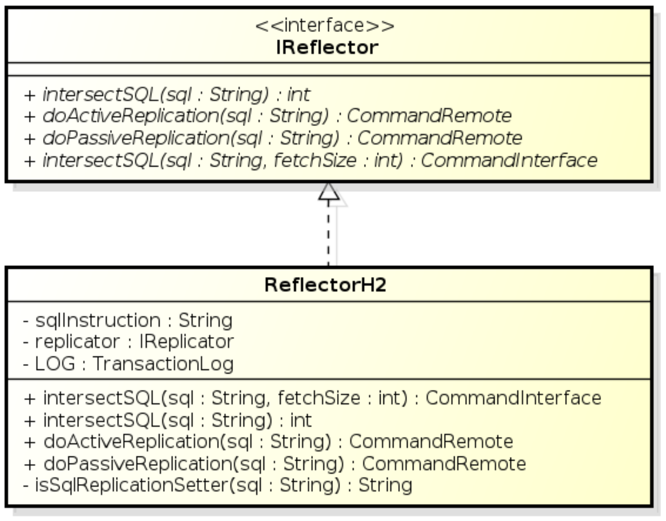
The recovery is done at the time of database startup. When the DBMS is started on the server, if a replication group already exists, the new DBMS that failed will join to this group. At this moment, begins the current state transmission and all the updates to be performed. This state is based on the database operation log – which is implemented in order to build the **recovery mechanism**. The log of failed node contains the last operation performed by this node. Thus, the leader receives it and compares with his last operation from his log, so that the leader can send the missing operations to the failed node.

The communication between the new member and the leader of the group during the recovery process is done via unicast. Upon receiving the instructions, the new member will execute them and update their status according to the leader's state. Accomplished this process, the new replica is now updated and ready to receive new transactions coming from the client.

We developed all replication mechanism for passive and active replication. The replication components implemented are described as follows.

**Reflector.** The Reflector component is based on architecture model presented in [2]. That component is a driver connected to the database engine and its role is to intercept all transactions directed to database engine and send them to Replicator. Reflector must to be aware of the replication strategy (i.e. if active or passive), so the proper replication method can be applied effectively.

Figure 8 shows Reflector interface, providing generic behavior for any kind of database system, and a specialized class to perform reflection on H2DB: ReflectorH2.



<<interface>>
IReflector
+ intersectSQL(sql : String) : int
+ doActiveReplication(sql : String) : CommandRemote
+ doPassiveReplication(sql : String) : CommandRemote
+ intersectSQL(sql : String, fetchSize : int) : CommandInterface
ReflectorH2
- sqlInstruction : String
- replicator : IReplicator
- LOG : TransactionLog
+ intersectSQL(sql : String, fetchSize : int) : CommandInterface
+ intersectSQL(sql : String) : int
+ doActiveReplication(sql : String) : CommandRemote
+ doPassiveReplication(sql : String) : CommandRemote
- isSqlReplicationSetter(sql : String) : String

Figure 8. Class diagram of Reflector component

The *intersectSQL* is a method that is called when the interception of SQL instructions in database transaction engine. This method makes the necessary treatments for the SQL instructions and verify which replication strategy will be used to perform the transaction.

The methods *doActiveReplication* and *doPassiveReplication* performs effectively the active and the passive replication technique, respectively. Depending on the strategy adopted, the Reflector can invoke the Replicator instantly or delay it.

The *isSqlReplicationSetter* is an auxiliary method to validate if the query is a specific SQL command recognized just by the database. This command is used to change the replication strategy at execution time and is detailed further in this section.

**Replicator.** Replicator is responsible for coordinating and interaction among all DBMS replicas, so it interacts with Reflector through well-defined interfaces and relies on the group communication component, as shown in Figure 9.

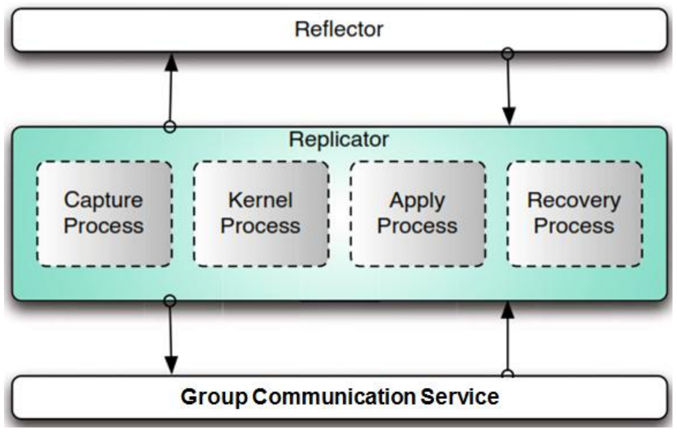


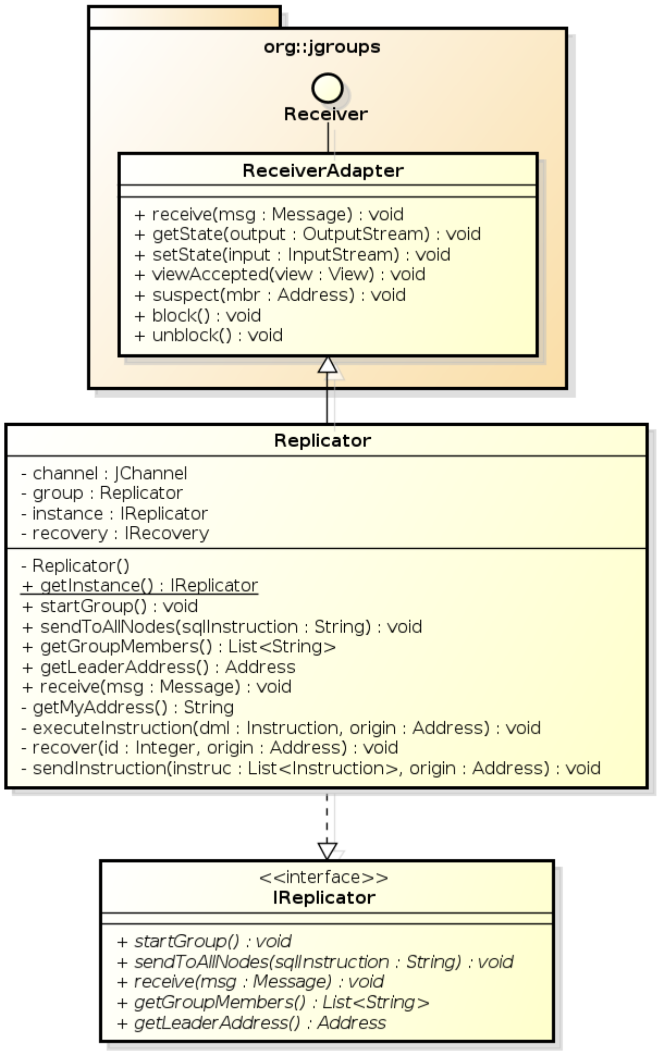
Figure 9. Replicator architecture. Adapted from [2]

There are four process abstraction built within Replicator component, presented as follows [2].

- **Capture Process** receives the events from Reflector, converts them to appropriate events within the replicator and notifies the other processes.
- **Kernel Process** handles the replication of local transactions by distributing relevant data and determining their global commit order. Additionally, it handles incoming data from remotely executed transactions.
- **Apply Process** injects incoming transaction updates into the local database through the reflector component. Thus, the communication between Reflector and Replicator is bilateral.
- **Recovery Process** intervenes and performs the recovery of some DBMS replica by other updated node. It is applied whenever a replica joins or rejoins the group.

Once the request (and the replication strategy to be used) is sent to Replicator, in the replication process, this can provide 1) replication strategy using group communication protocol, based on a generic interface; 2) a Replicator class; and 3) its interaction with group communication through Receiver (Figure 10). Receiver is based on JGroups toolkit, which provides total order multicast for group communication.

If the request is a write operation (e.g. update/insert command), Replicator send the transaction to the group; otherwise (i.e. a query command) it avoids to use group communication protocol and process the query locally in order to save network traffic. Figure 10 illustrates all the methods exposed by the Replicator that embraces the handling of group and the sending and receiving of write operations.



org::jgroups
Receiver
ReceiverAdapter
+ receive(msg : Message) : void
+ getState(output : OutputStream) : void
+ setState(input : InputStream) : void
+ viewAccepted(view : View) : void
+ suspect(mbr : Address) : void
+ block() : void
+ unblock() : void
Replicator
- channel : JChannel
- group : Replicator
- instance : IReplicator
- recovery : IRecovery
- Replicator()
+ getInstance() : IReplicator
+ startGroup() : void
+ sendToAllNodes(sqlInstruction : String) : void
+ getGroupMembers() : List<String>
+ getLeaderAddress() : Address
+ receive(msg : Message) : void
- getMyAddress() : String
- executeInstruction(dml : Instruction, origin : Address) : void
- recover(id : Integer, origin : Address) : void
- sendInstruction(instruc : List<Instruction>, origin : Address) : void
<<interface>>
IReplicator
+ startGroup() : void
+ sendToAllNodes(sqlInstruction : String) : void
+ receive(msg : Message) : void
+ getGroupMembers() : List<String>
+ getLeaderAddress() : Address

Figure 10. Class diagram of Replicator component with GC

The method *startGroup* instantiate a new group in the environment, if there is no group already created.

The *sendToAllNodes* operation receives the transaction and sends it by multicast to all members. Part of this method can be represented as the Capture Process of Replicator.

The method *receive* is part of group communication mechanism which receive the message (SQL transaction) sent by a node (replica). It is an implementation of the method in Receiver component. Both *receive* and part of *sendToAllNodes* methods might be represented as the Kernel Process of Replicator.

The *getLeaderAddress* is an auxiliary method. As the name suggests, it gives the address of the group leader (also called as primary). It is useful when some replica needs to perform the recovery process. In this case, the node obtains the leader address through this method and asks to the leader for the updates.

Additionally, the *getGroupMembers* is also an auxiliary method which returns all the active members within the group of replicas.

The method *executeInstruction* can be represented as the Apply Process of Replicator. It calls the Reflector again to execute the SQL instruction locally.

The method *recovery* verifies the necessity of performs recovery in a DBMS that joined in the group. It symbolizes the Recovery Process of Replicator.

Finally, to provide dynamic change of replication strategy, “Current Replication Strategy” component keeps the current replication strategy, queried by Reflector. The command bellow can change the replication strategy:

- **SET (ACTIVE | PASSIVE) REPLICATION;**

That command is provided as a special SQL command that allows the changing of current replication strategy. Reflector process that command and registers the new strategy on “Current Replication Strategy”.

## VI. Validation and Results

As seen in Section IV, H2DB is a relational and open source DBMS developed with Java programming language. To allow our performance evaluation, we have changed the H2DB architecture in order to use our middleware at transactions process engine.

As database servers have been relevant to business and have become more standardized over years, performance has been the differential factor among products of varied vendors. Performance benchmarks are bundles of tasks that are used to quantify the software performance so that people can measure quality, time and costs.

This project measures the transactions per minute as a performance indicator in order to assess the latter within the H2 database replicated environment. It is also applied in this project a benchmark tool used for measuring performance in database systems: TPC-C.

### A. *Description*

This section demonstrates all steps made to corroborate the new version of H2DB (supporting replication). In the end, there is a discussion of results comparing the both replication strategies adopted: active and passive.

Regarding validation of built implementation, the technique of TPC-C benchmark is used as a means of producing a specific load on the system in order to verify the effective compliance of the outlined objectives. Based on the results retrieved by the benchmarks, performance measurement from environment was made in the following three scenarios:

1) *Centralized*: with only a single H2 database storing data. In this scenario the benchmarks run on H2DB in just a single machine with the database switched in centralized mode.

2) *Synchronously replicated* (active replication mode): with two nodes in network supporting H2 database. In this scenario there is a replica of H2DB communicating with another replica on a different machine, yet in the same network. The communication between the replicas is done through group communication method using JGroups; the DBMSs are bounded in active replication mode.

3) *Asynchronously replicated* (passive replication mode): with two nodes in network supporting H2 database. In this scenario there is also a replica of H2DB communicating with another replica on a different machine, yet in the same network. The communication between the replicas is done through group communication method using JGroups; the DBMSs are bounded in passive replication mode.

Firstly, the environment features built for testing and validation are exposed in this Section. Subsequently, it is discussed the benchmark model (TPC-C) used and its peculiarities. After, it is presented the execution of scenarios with charts illustrating the results. Finally, from the results obtained, a comparative analysis of the different scenarios is taken in order to draw the conclusion.

### B. *Environment Features*

Because it is an essential factor in distributed and ubiquitous computing, heterogeneity and scalability are designed for the built environment. To perform the validation of the three scenarios defined, a physical machine and a virtual machine with different characteristics have been done towards greater similarity to real environments of distributed computing.

The environment is compound of a physical machine with virtual machine (VM) installed. The physical machine’s features are as follows:

- Dell machine with processor Intel® Core™ i7-3540M CPU @ 3.00GHz and 4MB Cache L3
- 8,0 GB of DDR3 Memory RAM
- 500 GB space of hard drive
- Windows 7 Operating System Professional x86 of 64-bit
- Network Adapter Intel® Centrino® Advanced-N 6205
- Java Virtual Machine 6.0 or greater installed
- VMWare Player 6.0.1 installed containing a virtual machine

The settings of each virtual machine inherited from the physical machine are as follows:

- Single-Core Processor
- 1,5 GB of DDR3 Memory RAM
- 8,0 GB space of hard drive
- Windows XP Operating System x64 of 32-bits

- Network adapter in Bridge mode

- Java Virtual Machine 6.0 or greater installed

The virtual machine installed and running under VMWare was used to simulate the replication over the synchronous strategy (active replication) and asynchronous strategy (passive replication). The network adapter connected and enabled in Bridge mode allows connection to the physical network and incorporates it in the same network of physical Windows 7 machine, as shown in Figure 11:

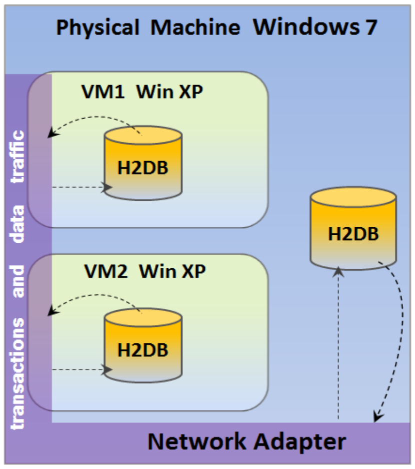


Figure 11. Architecture environment for validation

On each machine it has been deployed the new version of replicable H2DB, which has its own console server to database access and logs reading.

With regard to scalability, the environment also includes such a feature. If it is desirable to increase the number of replicas in the whole environment, it is possible to attach more nodes depending on the needs of a particular scenario. To do so, simply create and add a new virtual or physical machine, incorporating it into the same network of the other nodes, and deploy H2 database in replicable mode into the new machine.

Thus, the new replica will join the group and will be recognized as a new member - also applying the replication of transactions sent to all DBMSs. There is therefore a minimal effort to increase the number of replicas in the environment, supporting the concept of system horizontal scalability (scale out).

## *C. TPC-C Benchmark Model*

In order to perform benchmarks for the new version of H2DB, it was used a worldwide renowned patterns defined by a non-profit organization, namely, Transaction Processing Performance Council (TPC). The purpose of TPC is to define transaction processing and database benchmarks used by the industry in pursuance of evaluating the performance of computer systems [20].

Generally, the TPC creates benchmarks that assess transaction processing (TP) and database (DB) performance. These measures provide an inference of how many transactions a particular system or database can perform per unit of time. The evaluation of this project, specifically, measures the number of transactions per minute (tpm).

In the same way as tpm measure, the TPC presents and additional measuring entitled tpmC. This metric concern measures the New-Order transactions (type of TPC-C transaction which will be described forward) per minute and the number of orders as it can be fully processed per minute. During the execution of benchmarks, we compute the total number of performed New-Order transactions. Then, we divide this value by the subtraction of the time that execution finished and the time it was started. Given this result, we multiply by 60000 which is the absolute value, in milliseconds, of the desired unit (in this case, minute). Thus, the benchmark tool can reach an approximate measurement of tpmC. The equation below shows the formula to calculate the tpmC:

$$\boldsymbol{tpmC} = 60000 \times \frac{nrNO}{(endTime - initialTime)}$$

The *nrNO* represents the total number of New-Order transactions executed during the unit of time (one minute). The tpmC depends strictly on overall system characteristics – described previously. As a result, all values obtained in the benchmarks' execution of this project also include the tpmC metric and are related to the environment features adopted.

According to TPC standard, there are some types of benchmarks to support different needs, such as TPC-Energy (for measuring and reporting an energy metric in TPC Benchmarks), TPC-E (for On-Line Transaction Processing – OLTP – workload developed by the TPC),

TPC-H (for *ad hoc* decision support benchmark), TPC-C (for a simulation of complete computing environment where a population of users executes transactions against a database), among others that can be found in more details in [20].

As it is required for this project to make the measuring of transactions against a database, it was chosen the TPC-C approach [21]. This approach is a combination of read-only and update transactions that simulate the intense activities found in complex OLTP application environments. Moreover, the TPC-C model represents the activity of most common industry which must manage, sell or distribute a product or service, instead of focus in a particular industry activity.

Additionally, TPC-C has five types of transactions that are performed during the benchmark execution. These transactions have different characteristics, which can vary on weight (some are heavier than others) and, consequently, on the time spent to be performed in the database in order to simulate a production environment of data manipulation. Each transaction type belonging to TPC-C executed in all benchmarks scenarios of this project is described as follows [21].

**New-Order transaction.** This is a mid-weight transaction and it is considered as the backbone of the workload. It is comprised of data reads and writes which are frequently executed and has a stringent response time requirements to fulfill online users' necessities. As the name implies, the New-Order transaction consists of entering a complete business order through a single database transaction. It is designed to place a variable load on the system to reflect on-line database activity that meets to real environment actions.

A new order is done in a single transaction with the following steps:

1. Creation of an order header containing:
   - ✓ 2 row selections with data retrieval;
   - ✓ 1 row selection with data retrieval and update;
   - ✓ 2 row insertions.

2. Request a variable number of items (average *items_cnt* = 10) to be included in the order header created, containing:
   - ✓ (1 × *items_cnt*) row selections with data retrieval;
   - ✓ (1 × *items_cnt*) row selections with data retrieval and update;
   - ✓ (1 × *items_cnt*) row insertions.

**Payment transaction.** This is a light-weight transaction comprised of data reads and writes which are frequently executed. It has also a stringent response time requirements to fulfill online users' necessities, as it does not include primary key access to the table of customers. As the name implies, the Payment business transaction updates the customer's balance and reflects the payment on the district and warehouse sales statistics.

The Payment transaction enters a customer's payment with a single database transaction and performs about 3 or 5 selections and 1 row insertion.

**Order-Status transaction.** This is a mid-weight transaction comprised of read-only database transaction with a low frequency of execution. It has also a low response time requirements to fulfill online users' necessities, as it does not include primary key access to the table of customers. As the name implies, the Order-Status business transaction queries the status of a customer's last order.

The Order-Status transaction performs about 2 or 4 rows selections with data retrieval to find the customer; and 1 × *items_per_order* (average *items_per_order* = 10) rows selections to check the delivery date of each item on the order requested.

**Delivery transaction.** This type could be comprised of one up to 10 database transactions. It has a low frequency of execution and does not necessary need to be completed within a stringent response time as a requirement. This transaction consists of processing a batch of 10 new (not yet delivered) orders. Each order is processed (delivered) in full within the scope of a read-write database transaction.

The Delivery transaction delivers one outstanding order (average items-per-order = 10) for each district in the warehouse using one or more (up to 10) database transaction. This process includes, for each order: 1 row selection with data retrieval; 1 row selection with data update; and 1 row deletion.

**Stock-Level transaction.** This is a heavy read-only database transaction with a low frequency of execution. It does not necessary need to be completed within a stringent response time as a requirement and has relaxed consistency requirements. This transaction determines the number of recently sold items that have a stock level below a specified threshold.

As seen, the TPC concerns a commonly understood group of transactions in the business sphere, such as commercial exchange of goods, services or money. Hence, the transactions would include the updating to a database system for an inventory control (goods), airline reservations (services), banking system (money), among others. Furthermore the TPC-C model addresses the basic needs for measuring performance on database systems, including patterns and a set of different transaction types. For this reason, we have been using the TPC-C as a means to measure performance over H2 database in replication approaches and even in centralized mode.

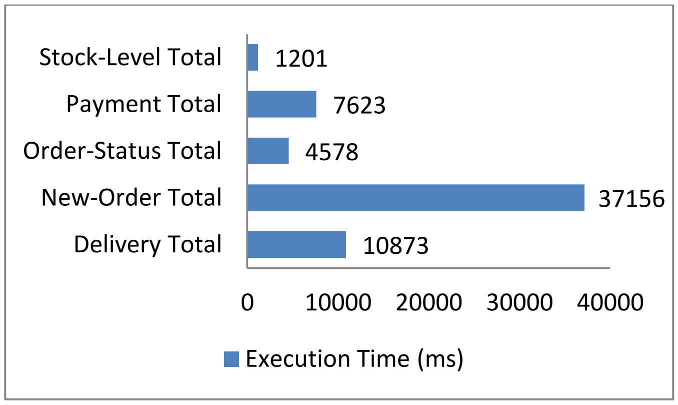
### D. *Execution of Scenarios and Results*

The benchmarks were executed according the scenarios described previously. The database tables required by TPC-C model [21] were created in H2DB and then we populated the tables with 598.587 rows. The time taken for each benchmark was 60.000 milliseconds. Moreover, for each scenario was opened a terminal where all transaction types (New-Order, Payment, Order-Status, Delivery and Stock-Level) were performed during 1 minute (60.000 milliseconds).

We have considered, in percentage, the following transaction weight:

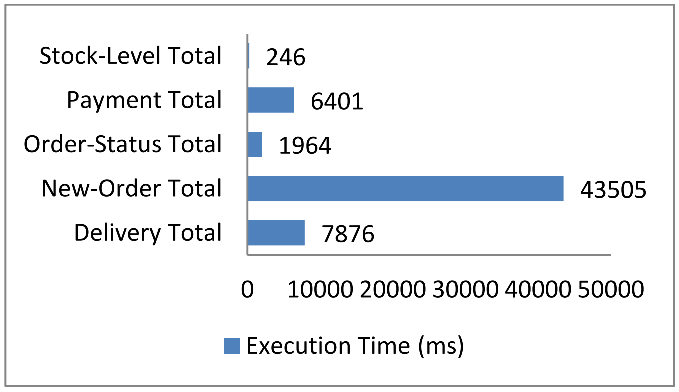
- New-Order: 45%
- Payment : 43%
- Order-Status: 4%
- Delivery: 4%
- Stock-Level: 4%

In centralized mode, a range of transactions were executed over H2DB in order to measure the total time that each transaction type took to be fully processed in the DBMS. Graphic1 illustrates the results of these measurements:



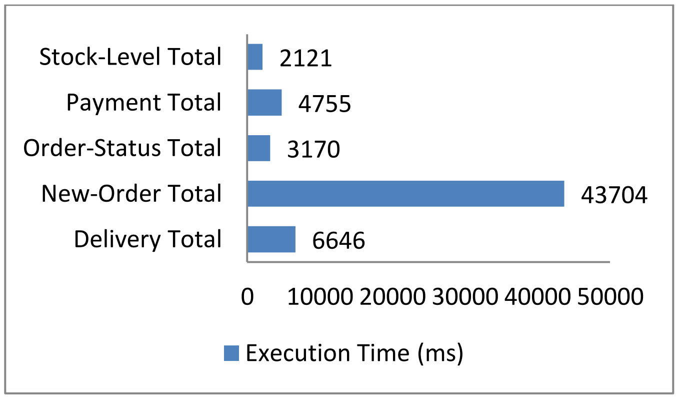
Graphic1. Benchmark results: Centralized

In passive replication the database was switched to passive mode. Then we opened a new terminal to run the benchmarks. We have also measured the total time of each transaction type – illustrated in Graphic2.



Graphic2. Benchmark results: Passive Replication

Finally, active replication was switched in H2DB. Then we also opened a new terminal to run the benchmarks and measure the total time of each transaction type, as shown in Graphic 3.



Graphic 3.Benchmark results: Active Replication

Because the New-Order transaction is a mid-weight transaction and it is considered as the backbone of the workload, it spends more time executing than the others. Even having similar weight to the Payment transaction (45% for New-Order and 43% for Payment), the New-Order lasted a significantly time longer. It is explained because Payment transaction is a light-weight transaction.

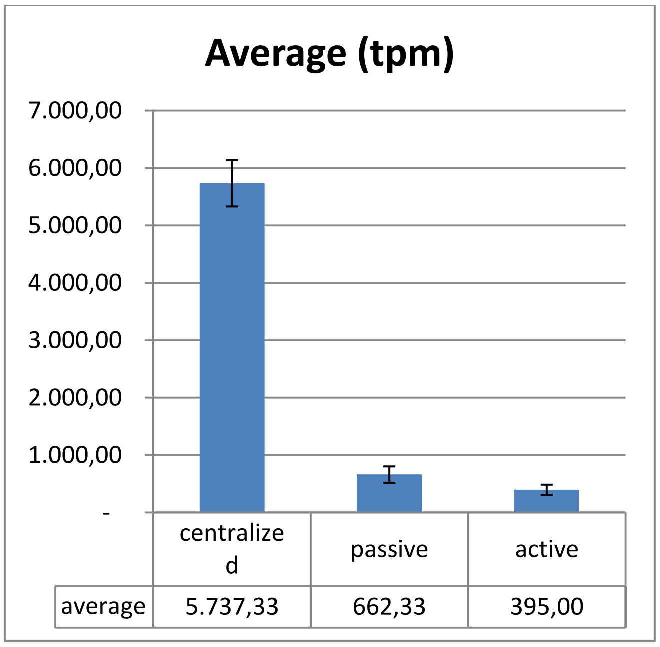
Based on TPC-C Ben chmarks executed over H2DB in centralized and replicable mode, we can demonstrate the behavior of different scenarios and evaluate the results. As expected, the centralized database could handle a significantly larger number of transactions compared with the same environment and database in replicable mode. Furthermore, the passive replication performed more transactions than active replication during the same period of time.

Three benchmarks execution were done for each database mode. Table 2 shows all results obtained in transactions per minute (tpm).

| | Centralized | Passive Rep. | Active Rep. |
|---|---|---|---|
| Benchmark 1 | 5.381 tpm | 783 tpm | 321 tpm |
| Benchmark 2 | 5.736 tpm | 673 tpm | 482 tpm |
| Benchmark 3 | 6.095 tpm | 531 tpm | 382 tpm |
| Average | 5.737,33 | 662,33 | 395,00 |
| Standard Derivation | 357,00 | 126,34 | 81,28 |
| Confidence Interval | 403,98 | 142,96 | 91,98 |

Table 2. Results of benchmarks execution

We calculated the average tpm for each strategy in order to find the standard derivation and the confidence interval of 95%. From values extracted we can conclude that in 95% of benchmarks that could be executed in active mode, the transaction per minute can vary from 303tpm to 487tpm. On the other hand, the passive replication can vary from 519 to 805. Graphic 4 shows the average of transaction per minute for each strategy and the error bar according to the confidence interval calculated.



Graphic 4. Confidence interval of average results

We conclude that if it is necessary to execute more transactions (within a period of time) and give a faster reply to the client, our replicated database can be set up to apply the passive replication, since this is the replication strategy which executes more transactions per minute – as shown in Graphic 4. The passive replication however has a longer response time for failures when compared to active replication.

On the other hand, if it is needed an environment that all replicas are 100% of time synchronized and the clients are able to send requests to any replica (update anywhere model, presented previously), we can switch at runtime to set the active replication with no effort, as described in Section IV.

## VII. Conclusions And Future Work

This work aims at exploring an approach for adaptability in the context of replicable database environments. It shows that database replication is a complex solution that involves different decisions to every particular problem. Although it might be hard and expensive to apply, if developed in the right circumstances for a particular issue, the replication can be fitted perfectly as a good solution, enhancing both performance and high availability for fault tolerant systems and scenarios that require high scalability.

Thus, this project encompasses the developing of a middleware, for database replication, based on a generic architecture and adaptable to different replication strategies. Figure 12 illustrates the scheme proposed for a cloud environment within the replication middleware.

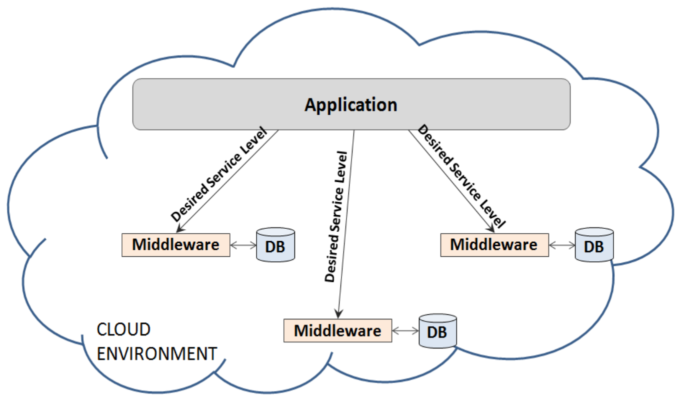


Figure 12. Proposed scheme for a cloud environment

As shown in Figure 12, the application can define the desired service level for database nodes. This means that an application is allowed to switch at runtime the replication level. If it is suitable for the application to have a passive environment at a certain instant, application can send an operation to database middleware defining the passive mode. On the other hand, there could be another situation that is preferable to have an active environment, so the application is able to interact with the middleware which can change it at execution time.

Furthermore, some benchmarks based on TPC-C [21] model were executed over the replicated database environment in order to validate the implementation. As a result, we calculated the tpm (transactions per minute) and obtained the average, standard derivation and confidence interval. Some charts were drawn to illustrate the results and compare them to each strategy that was implemented. The experimental results reported here point out that the building of a replication environment in H2DB is consistent and the processing times correspond to what was expected – according to the foundation studies of replication strategies.

Finally, a proposal for future works is an implementation focused on commercial environment of cloud computing which the choice of replication strategy is not regarding to the application that uses the database or any other user (such as a database administrator). Instead, the cloud environment should be able to adjust automatically the database replication strategy based on the computational resources measuring. There should be a component whose role is to measure a set of resources that might impact the performance of the whole system, such as:

- CPU time;
- Memory usage;
- Hard disk space;
- Network throughput;
- Input/output operations;
- Electrical power, etc.

For instance, if there are plenty of available resources, the cloud environment can set the active replication; if not, it can switch for passive replication automatically in order to minimize the use of network resources and, moreover, use fewer number of replica nodes – since this strategy requires less resource consumption.

## References

[1] “*The Information Society. From Theory to Political Practice*”. Information Society Research Institute, First edition. Gondolat Kiadó, Budapeste, 2008.

[2] A.T.C. Jr., J. Pereira, L. Rodrigues, N. Carvalho, and R. Oliveira. *Practical Database Replication*. Replication Theory and Practice, B. Charron-Bost, F. Pedone, A. Schiper, Cap. 13. Springer, 2010.

[3] M. Armbrust , A. Fox , R. Griffith , A. D. Joseph , R. Katz , A. Konwinski , G. Lee , D. Patterson , A. Rabkin , I. Stoica , M. Zaharia. *A view of cloud computing*. Communications of the ACM, v.53 n.4, April 2010

[4] X. Defago, A. Schiper, and Peter Urban. *Total order broadcast and multicast algorithms: Taxonomy and survey.*ACM ComputingSurveys, Dec 2004.

[5] P. Tomar,Megha. *An Overview of Distributed Databases*. International Journal of Information and Computation Technology. International Research Publications House. Volume 4, Number 2, p. 207-214, 2014.

[6] R. Elmasri, S. B. Navathe. *Fundamentals of Database Systems*. Pearson. 6 ed. 2010.

[7] A.T.C. Jr. *Practical Database Replication*, Universidade do Minho, Escola de Engenharia, Portugal, 2010.

[8] R. Renesse and R. Guerraoui. *Replication Techniques for Availability*. Replication Theory and Practice, B.

Charron-Bost, F. Pedone, A. Schiper, Cap. 2. Springer, 2010.

[9] J. Gray, P. Helland, P. O' Neil, D. Shasha. *The dangers of replication and a solution*. ACM SIGMOD International Conference on Management of Data, p. 173-182, June 1996, Montreal, Quebec, Canada.

[10] Oracle Corporation. *Database Replication and Integration*. Available at http://www.oracle.com/technetwork/database/features/data-integration/index.html

[11] Microsoft Corporation. *SQL Server Replication*. Available at http://technet.microsoft.com/en-us/library/ms151198.aspx

[12] P. Jalote,*Fault Tolerance in Distributed Systems*, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[13] A. Avizienis, J.C. Laprie, B.Randell, and C.Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, pp. 11-33, Jan 2004.

[14] A. Schiper, *Group communication: from practice to theory*, Proceedings of the 32nd conference on Current Trends in Theory and Practice of Computer Science, 2006, Měřín, Czech Republic

[15] G. Samaras, K. Britton, A. Citron, C. Mohan. *Two-Phase Commit Optimizations and Tradeoffs in the Commercial Environment*, Proceedings of the Ninth International Conference on Data Engineering, p.520-529, April 1993

[16] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. *Understanding Replication in Databases and Distributed Systems*. In IEEE ICDCS, 2000

[17] JBoss Application Server by Redhat. Available at http://jbossas.jboss.org

[18] H2 Database Engine. Available at http://h2database.com

[19] JGroups – A Toolkit for Reliable Messaging by Redhat. Available at http://www.jgroups.org

[20] TPC – Transaction Processing Performance Council ™. Available at http://www.tpc.org

[21] TPC BENCHMARK™ C Standard Specification Revision 5.11, February 2010. Available at http://www.tpc.org/tpcc/spec/tpcc_current.pdf

**Bruno Cesar Neves de Oliveira** received his bachelor degree of Computer Science at UNIFACS – University of Salvador in 2011. He works in the field of system's development as a System Analyst since 2009. Currently, works at Informática El Corte Inglés in Salvador, Bahia, Brazil and is attending the post-graduation course of Specialization in Distributed and Ubiquitous Computing at Federal Institute of Bahia (IFBA).