CZECH TECHNICAL UNIVERSITY IN PRAGUE FACULTY OF INFORMATION TECHNOLOGY DEPARTMENT OF SOFTWARE ENGINEERING

Bachelor's thesis

Synchronized Database Storage

Sanan Pashayev Bachelor

Supervisor: Ing. Tomáš Hradský

08th July 2019

Acknowledgements

I want to thank my supervisor Ing. Tomáš Hradský for his invaluable assistance during writing this thesis.

Above all, I would like to thank my family for supporting and motivating me during all the time.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

on 8th July 2019 .....

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

Sanan Pashayev. Synchronized Database Storage. Czech Technical University in Prague, Bachelor's thesis. Faculty of Information Technology, 2019.

Abstract

Data synchronization ensure that a data set exists in different directory servers to ensure redundancy as well as faster processing. Such servers need to have data that is consistent and available at all times. Data consistency and availability can be assured by data synchronization and data replication. Data replication involves creation of more than one redundant database copies to improve on data accessibility as well as make the system fault tolerant.

This thesis is intended to present a study a working tool for database synchronization without forgetting to look at conflict introduced by this duplication of information and their resolution. An algorithm that works to ensure synchronization using a client-side tool is used to present a working model. This study is based on experiments on real data to show how the synchronization techniques can solve the data inconsistencies and guarantee the integrity of the data.

Keywords

Database Synchronization, offline data, conflict resolution, SQLite, PostgreSql, data integrity, synchronization techniques.

Abstrakt

Synchronizace dat umožňuje existence stejných dat na různých serverech. Tím je zajištěna redundance a rychlejšího zpracování. Tyto servery musí mít konzistentní a neustále dostupné údaje. Konzistenci a dostupnost dat lze zajistit synchronizací dat a replikací dat. Replikace dat zahrnuje vytvoření více než jedné redundantní kopie databáze ke zlepšení dostupnosti dat a také k tomu, aby byl systém odolný vůči chybám.

Cílem této práce je představit studii, která by fungovala jako nástroj pro synchronizaci databází, s ohledem na konflikty, který tato duplikace informací může zapříčinit. Je použit algoritmus který zajišťuje synchronizaci pomocí nástroje na straně klienta. Tato studie je založena na experimentech na reálných datech, které ukazují, jak mohou synchronizační techniky řešit nesrovnalosti dat a zaručita jejich integritu.

Klíčová slova

Databázová synchronizace, offline data, řešení konfliktů, SQLite, PostgreSql, integrita dat, synchronizační techniky.

Table of Contents

Table of Figures

1. Introduction

The world of technology has experienced great advancements which have major impacts on our lives. This is well manifested through our day-to-day activities which involve interactions with the internet. Our modern society is characterized by production and consumption of huge gigabytes of information every single day. One type of information includes news which can be accessed through various media such as TV, newspaper, website or even social media. All this information is stored in databases where it can be accessed by various people upon request. Therefore, this leads to the creation of databases which can be considered to be readily available around us. The effects related to these databases are extensive and we experience them in almost all of our daily activities. For instance, weather applications, online movie streaming, social gaming, and cloud storage are some of the activities which involve the application of databases. In addition, the use of databases has become a crucial element and necessity for organizations in order to promote the sharing of information in a seamless manner. This helps in improving the quality and availability of data. Data availability is achieved through the use of distributed databases technique which refers to a collection of databases that are shared over the network. Data sets are made available and consistent in more than one database. In order to ensure this process is implemented successfully, both data replication and synchronization processes are performed. Data replication refers to the process of creating more than one redundant copy of a database to ensure accessibility improvement as well as fault tolerance. On the other hand, data synchronization refers to the process of establishing consistency of data across all databases even after subsequent updates.

Database synchronization has led to huge advancements of database systems by promoting parallel data processing. This has led to major improvements which have enabled real-time accessibility of data to many people across the world. Users are able to use multiple devices that are always in sync, and real-time experience is guaranteed. Database synchronization is achieved through the use of a specialized tool that keeps track of database versions as they are created and utilized. When a change is detected in one database, all the other nodes are notified and they update accordingly. The database synchronization process can be implemented manually by making use of the Microsoft Sync Framework. Alternatively, this can be achieved through the use of already existing inbuilt tools which come with systems such as Microsoft SQL Server.

This paper aims at developing a comprehensive analysis of database synchronization and all the necessary requirements that should be met. There is a short review of tools which have been already developed in order to support database synchronization. This exploration will help in establishing an existing gap that will form the research problem of this thesis. This paper aims to address and develop a suitable and viable solution. This is further followed by comprehensive documentation of the proposed software solution. There will be a detailed discussion on how that prototype has been developed as well as how it works in order to address the research problem. In addition, there will be documentation of the testing process and discussion of the benefits associated with that product.

1.1 Problem Statement

An environment which is associated with computational processes is characterized by use of client-server systems. In some instances, an offline application needs to interact with a server. This interaction involves sending and receiving of information that aims at ensuring all devices and the server information is synchronized. Additionally, a client software that is installed on a local database needs to be in sync with its server counterpart. Which mechanism can be adopted in order to ensure that data and structural changes have been handled correctly and all endpoints have updated data? This forms one of the major parts of the

problem statement that will be addressed in this paper. This thesis aims at developing a solution that will assist in full implementation and realization of database synchronization. The developed solution will be fully documented and test processes will be carried out in order to establish whether the problem statement is addressed.

1.2 Objectives

The main objective of this paper is to develop a database synchronization tool that will help in performing a two-way data synchronization between the server and local databases where client systems reside. This is achieved by ensuring that all devices within the client-server environment have updated data at all times. This paper will show how the database synchronization is developed and implemented in order to form a working model. Experiments using real data will be performed with an aim of showing how the tool works towards eliminating data inconsistencies and guarantee maintenance of data integrity at all times.

2. Related Work

There is a lot of work which has been done on this research topic with an aim of establishing a tool that could aid in database synchronization. A review of these tools is very crucial in this study because it will help in gathering knowledge that will assist in creating a suitable and effective data synchronization tool. Furthermore, reviewing these tools helps in identifying gaps which should be addressed by the proposed solution. Below are some of the existing database synchronization tools which were reviewed when preparing this paper.

2.1 Data synchronization using cloud storage (2012)

In accordance with (Anand 2012), they denoted that cloud computing devices comprised of back-end cloud servers and front-end cloud devices. This allows users to have an access to large volume of storage within the cloud architecture. In this paper, users can upload files from their devices such as mobile phones or desktops and send them to the cloud. This process is followed by synchronization of all files in all devices of the user that are connected to the internet. As a result, the user's files can be viewed from anywhere using any device. This service was created in order to provide a capability to the users that could allow them to upload and back-up their data from mobile devices to the SkyDrive. This could be done wirelessly without involving use of a Zune software.

2.1.1 System Architecture

In order to understand about how this solution was implemented, below is an overview of its system architecture.

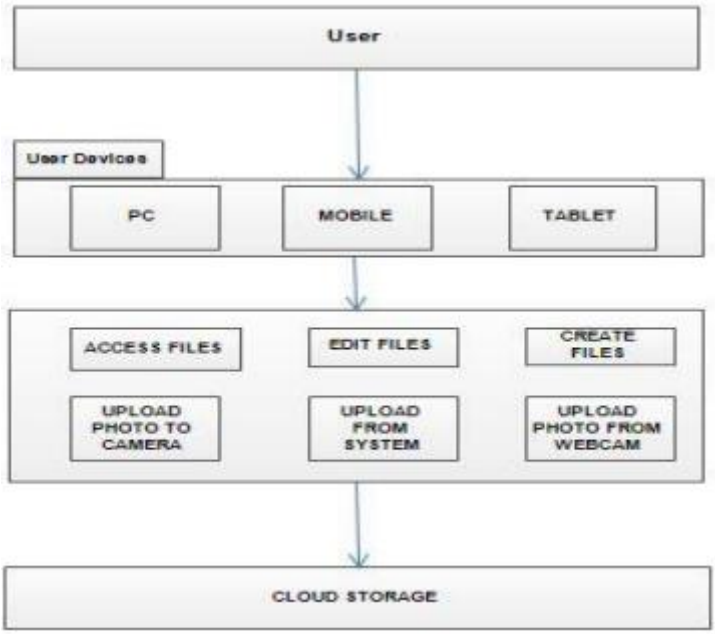

Figure 1: Cloud Data synchronization

This architecture comprises of user interface, user devices and a user. User refers to the person who will be interacting with the user interface while the user device refers to the platform which will be used by the user in order to access data. The user interface is used in accessing data and carrying data modification. In this cloud solution, the user is able to create a file a picture which could even be taken directly from directly from a webcam or camera. All windows platform has this interface implemented in them and whenever a file is uploaded or modified, synchronization takes place in all other devices. These files are transmitted into a cloud database where they can be accessed anywhere and from any device. For instance, in this solution, Sky drive is used as the cloud storage. Before accessing this service, a user is supposed to get an authentication via the user interface after which he or she can be able to carry out data editing and modification.

2.1.2 Implementation

The implementation process is broken down into the following processes as discussed below:

- ***Mobile Application***
  - ***Authentication***

    This is the initial process where a user is supposed to login into the Sky Drive platform using his or her login credentials. This process checks whether the user is valid and if not, the authentication fails. The figure below shows the login interface.

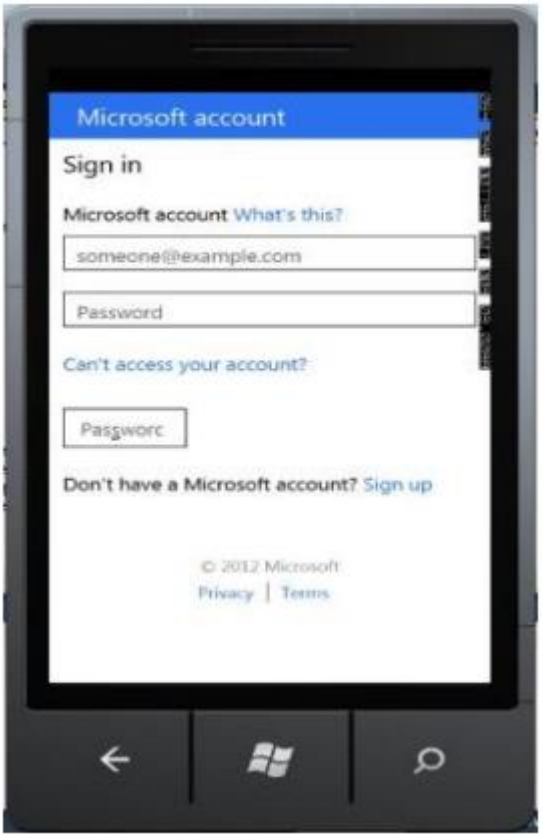

Figure 2: Mobile application UI

- Uploading photos from the camera
  This process is made possible through provision of an interface which enables a user to be able to upload a photo directly from the camera app in a Windows phone as well as Webcam from Windows desktop. The photos are uploaded to the SkyDrive and this helps in reducing the memory space of the user and grants access from any device and anywhere.
- Creation of Notes and uploading them
  There is an interface which enables the user to create some notes which can be uploaded to the SkyDrive. This interface also offers the capability of executing some changes. One can also be able to edit some existing documents such as DOC and PPT. Below is a figure showing this interface.

Figure 3: Mobile UI

- ***Desktop Application***
  - ***Photo hub***

    This is a desktop interface which is provided to the user by devices such as tablet, pc and laptop. This provides a view of all photos which have been uploaded into the SkyDrive and there is provision of a feature which will enable one to upload photos stored in the device or taken via webcam. Below is an overview of this interface:

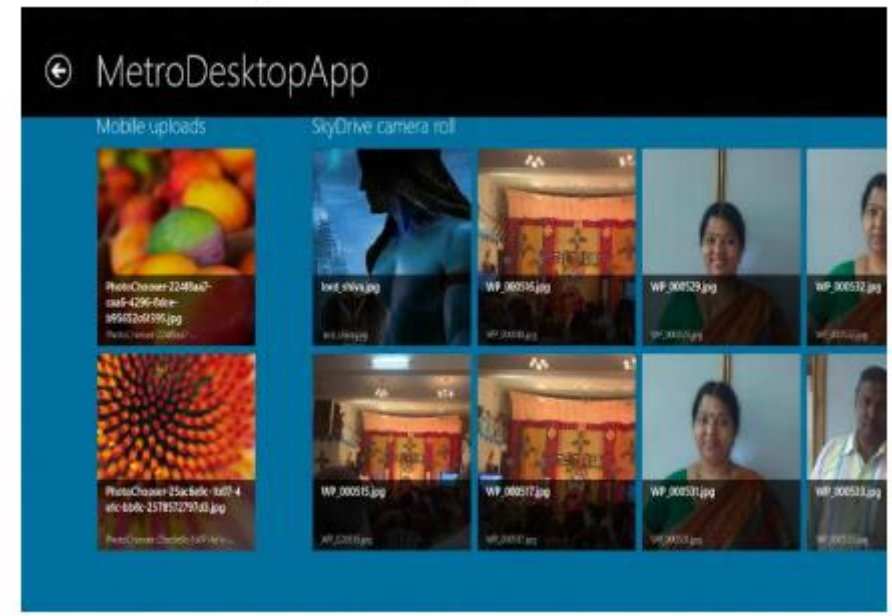

Figure 4: Desktop UI

- ***Document hub-***

  This interface is capable of displaying all documents which have various extensions such as .ppt, .pptx, .doc. dcox, .txt among others. The user is able to edit or create data new files which can then be uploaded into the SkyDrive as shown in the diagram below:

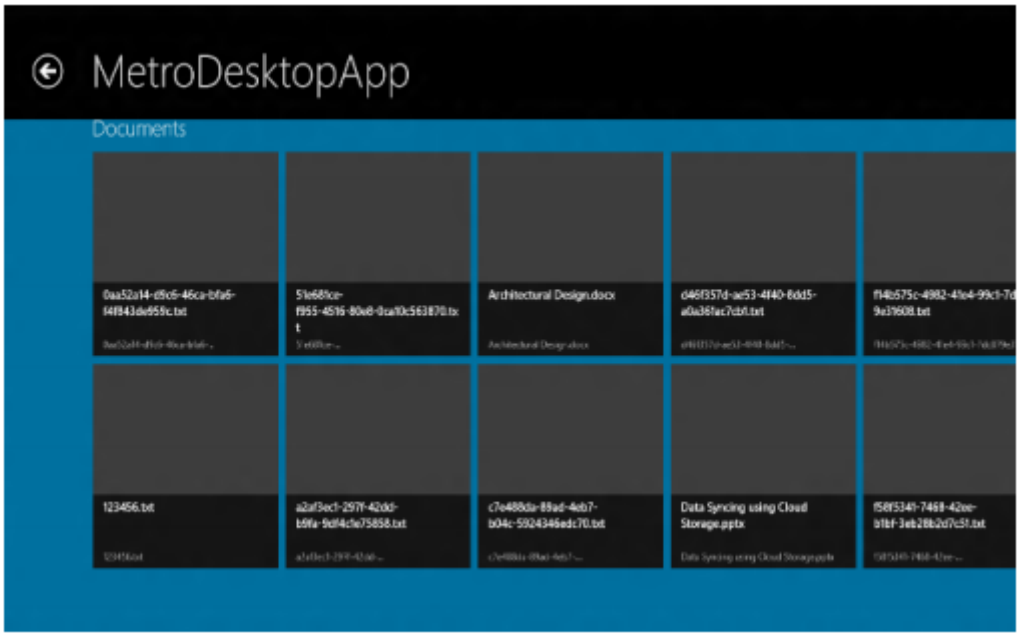

Figure 5: Desktop UI II

2.1.3 Discussion and Conclusion

By the use of credentials, user authentication process is carried out and if the process is unsuccessful, the user is denied an access to the SkyDrive account. If the authentication process is successful, the user is able to login and can be able to carry out data changes or modifications. Any changes on the data stored in the cloud is synced automatically and the user is guaranteed an access of updated data regardless of the device he or she is using when accessing the data.

2.2 Using data synchronization process as a way of solving problems related with software in absence of Network (2012)

According to authors (Betim Cico, Agni Dika and Isak Shabani) in their study [4], presented an algorithm that was applicable in carrying out data synchronization on Web Services (WS). This was essential in allowing software applications to work well on both online and offline configurations in an environment characterized by absence of network. This algorithm was implemented in University of Prishtina (UP) when developing Electronic Student Management System (ESMS). This was opted in order to address shortcomings that were as a result of uncertain power supply which kept on affecting network connection. Additionally, other reasons which affected internet connection and could not be solved by its professional staff, made this move more favorable to the university. Hence using this algorithm, it is possible to work offline and then data synchronization is done once connected to the network.

2.2.1 System Architecture

The main idea behind the development of this ESMS system was to create a solution that could support offline mode of operation. This system consisted of a central database and other 17 local databases which were used to store data for all faculties of this institution. The first step involved supplying all faculties of University of Philippines (UP) with Web servers and databases as shown below.

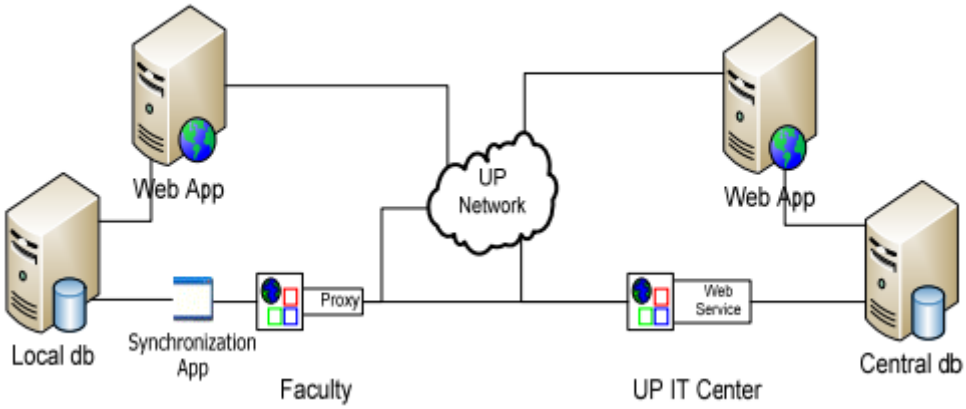

Figure 6: ESMS system

This was followed by installation of applications in all databases and web servers. This process ensured that all faculties had the systems at their disposal. However, the main challenge on how data could be synchronized between one side to another. In order to solve this problem, a special software system was developed with an aim of aiding in data synchronization between the central server and the other

local databases which resided in UP faculties. The following technologies were used in development of ESMS

- Server Operating system- Microsoft Windows Server 2008
- Client Operating system- Windows, mobile (Android) and Linux
- Development platform- .NET Framework 3.5
- Database- MS SQL Server 2008
- Web Server- MS IIS 7
- Browser- Mozilla Firefox, Opera, Safari, Internet Explorer
- Programming languages- C#, Ajax, Html, CSS, JavaScript, ASP.NET
- Development tools- Microsoft Visual Studio .Net 2008
- Accessing Data- ADO.NET
- Communication with another app- XML, Web Service

Algorithm for database synchronization

As a result of challenges which have been experienced by UP in ensuring data synchronization between all databases, the following algorithm was devised in order to address these challenges. This algorithm would help in ensuring that offline mode of operation was achieved and data across all databases was synchronized. On the basis of web services as shown in the diagram below.

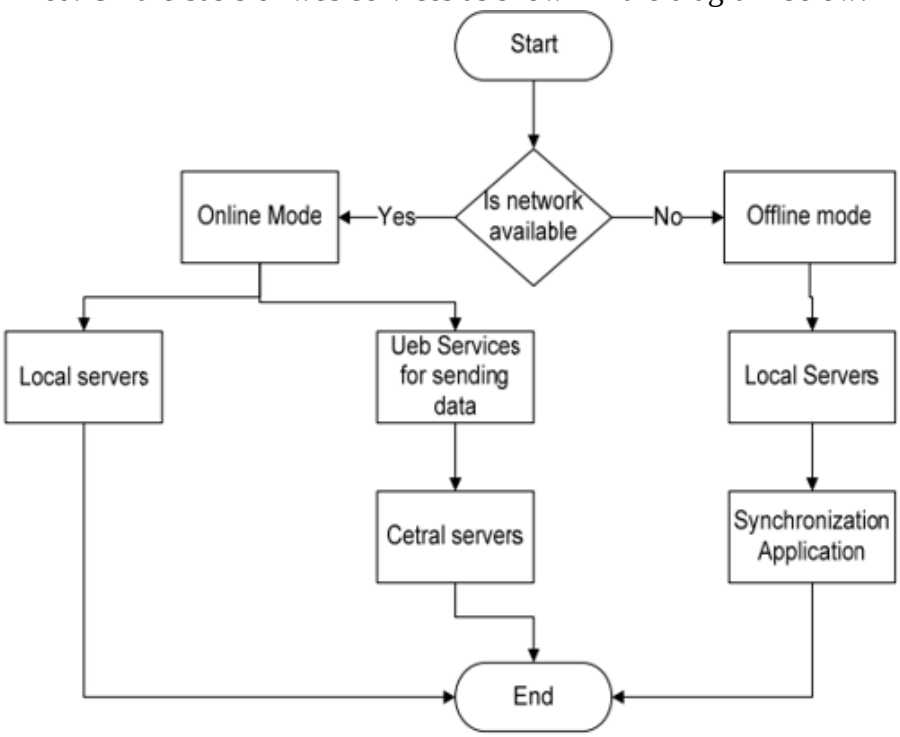

Figure 7: Database sync Algorithm

2.2.2 Implementation of the proposed solution for ESMS offline mode of operation

In order to achieve offline mode of operation, the proposed algorithm was implemented in an ESMS system. This enabled synchronization of data which between the faculties which were working offline together with the primary system. This was the main database and it could help in ensuring that all data was up to date with work that was done in various different units. WSs techniques were mostly used in the synchronization process. Different kinds of units were linked through a connection that was achieved

after creating a Proxy client. The main role of the proxy clients was to initiate WS methods that could be used to transfer data which had been stored offline as well as receive data from the real database.

2.3 Data synchronization in Driving Simulator-based Experiments for cognitive load estimation

In this study [5] as written by Andrew L. Kun and Zeljko Medenica, they present an analysis of effects which result from inattention and distraction on cognitive load. This has become a crucial factor due to the increased number of electronic devices that are finding their way towards the vehicle industry. Some equipment has been used in collection of different variables which are sensitive to changes which are associated with cognitive load. In order to obtain credible results as well as reliable conclusions, there need to be dependable ways of performing data synchronization from the various data sources. This paper presented a low-cost solution which helped in synchronizing three types of devices involved in driving research. This include physiological monitor, eye tracker and driving simulator (Medenica & Kun 2012).

.

2.3.1 System architecture

In order to understand how data synchronization was achieved, below is a look on both the hardware and software that was used.

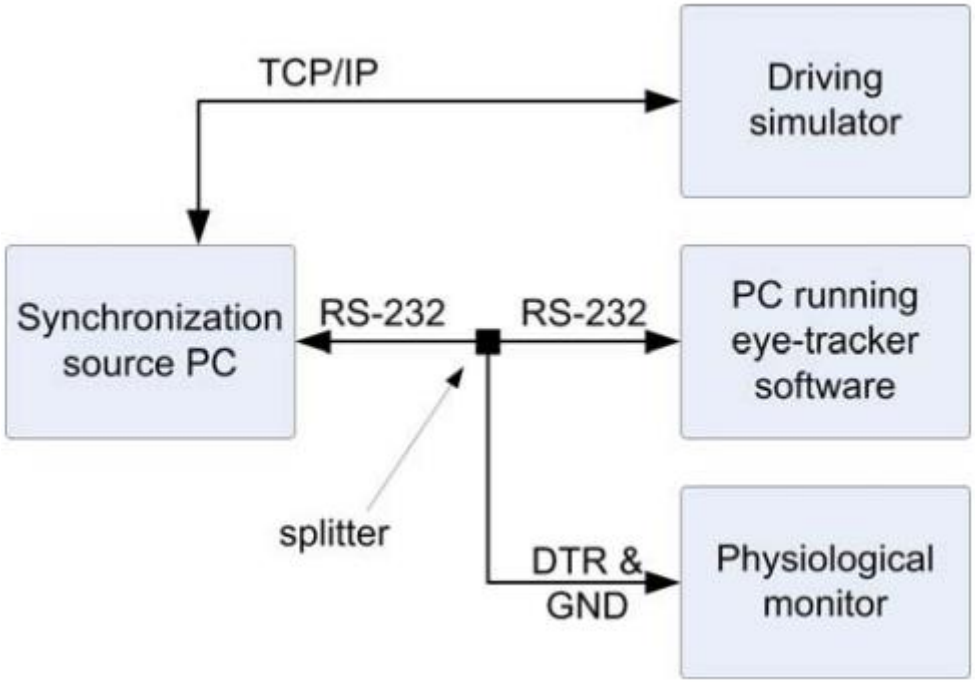

Figure 8: Synchronization concept

The first element is a PC which is the source of all synchronization messages which are initiated by the experimenter and sent to other equipment simultaneously. In this case, the PC ran on Microsoft Windows XP but other operating systems can be used as long as they can support both Serial (RS-232)

and TCP/IP communication. The following paths of communication were established between different equipment and the source PC. This is well discussed below:

2.3.1.1 Hardware side

- Driving simulator and TCP/IP communication path was established. The communication used by the local network could support high speeds up to 100mb/s. This approach could be extended up to other driving simulators which were PC-based. However, depending on the desired capability, serial or TCP/IP communication could be opted in this process.
- There was a serial communication path between the eye tracker software was being ran in a PC.
- The physiological monitor had a modified serial communication. This was essential in enabling sampling of multiple physiological signals in a simultaneous manner.

2.3.1.2 Software Side

There were different methods of communication which were used by all pieces of equipment which were used in this system. Hence, it was established that all synchronization messages could only be sent by the sources PC. This was crucial in giving attaching a timestamp to all synchronization messages depending on the time they were received. This led to the design of a customized application implemented in C++ that could help in realization of data synchronization. This application was responsible for sending the following messages.

- "SYNC" messages which were specified to the TCP/IP port of the driving simulator. The scripting system of the simulator could periodically screen all incoming messages and if the word "SYNC" was detected, the database of the driving simulator could acknowledge receipt of that message.
- Symbol "s" message which was transmitted to the eye tracker via the RS-232 port. A separate thread was chosen in order to ensure that the synchronization message was received as soon as possible. So, the blocking read call thread was chosen to perform this task. This meant that the application had to switch to a listening mode in order to detect this kind of a message. Once the "s" message is received, the local time is read and an update is carried on the log file.
- The PC had a serial port which could be toggled from low to high voltage at interval of about 0.5 seconds. This was crucial because the physiological sample monitor used the voltage levels in order to monitor changes associated with the samples recived.

2.4 A distributed Architecture in Distributed Database Systems for Transaction Processing (2010)

This is a paper [6] by Dr. Ajay Agarwal and Arun Kumar which gave a detailed description of proposed concurrency algorithms that could be used in distributed database systems. In order to achieve concurrency control of a distributed system, the available algorithms come in three basic classes. These are optimistic (or certification), timestamp and locking algorithms. This paper presented ways by which concurrency control is achieved in a distributed database system by using a Distributed Transaction Processing Model approach. This involved a breakdown of the problem of concurrency control into two main sub-problems namely; write-write and read-write synchronization. In this paper, there was a description of synchronization technique that solves each sub-problem and later these techniques are

combined into solving an entire problem. Such problems are referred to as concurrency control methods (Özsu, & Valduriez 2011).

In order to understand how a concurrency control algorithm is implemented in this kind of a database model, it is important to understand how the DDBMS works and how the algorithm fits into that kind of structure. DDBMS refers to a collection of sites which are all connected by use of network. It comprises of main components which are transactions, Transaction Managers (TMs), Data Managers (DMs) and Data. Communication of events flows from one component to another as shown below:

Transaction-Transaction Managers (TM)-Data Managers (DM)- Data

However, it is worth noting that there is no communication which take place between TMs with other TMs as well as DMs with other DMs. Below is a figure showing a DDBMS transaction processing model.

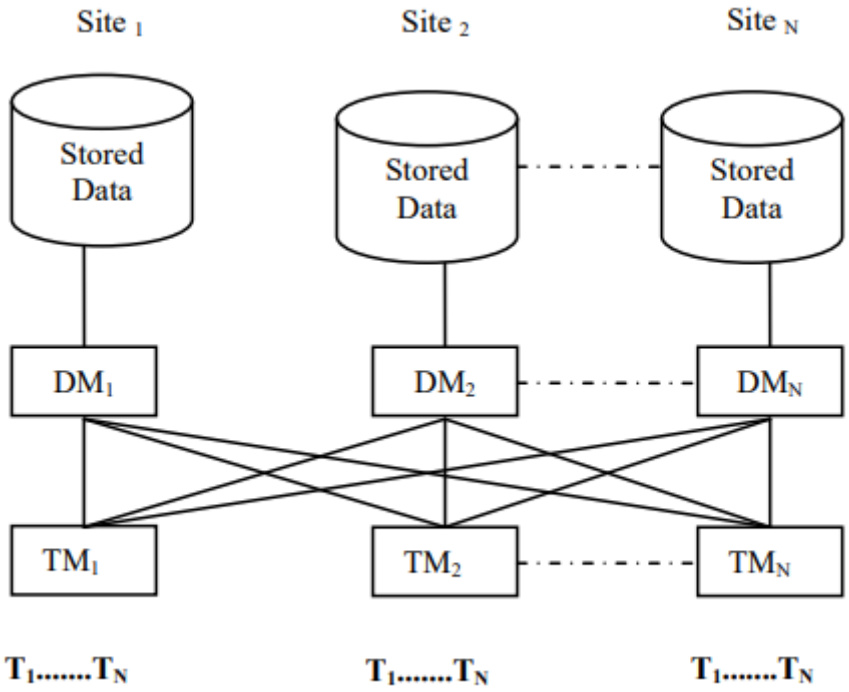

Figure 9: Distributed database transaction

A single TM is responsible for managing a specific transaction which takes place in a DDBMS. Hence, all transaction issues related to a database are submitted to a TM. TM is also tasked with taking charge of monitoring any distributed computation method that could be needed in execution of a transaction. Interactions between users and the DBMS are also supervised by the TM while the actual database is managed by the DMs.

In order to ensure there is data concurrency between all database in the DDBMS, two-phase locking technique is widely used. It achieves database synchronization by ensuring that read and write operations are detected explicitly in order to ensure that concurrent operations do not collide or cause conflict. This is achieved through creation of "lock" which must be obtained before carrying any kind

of operation. For instance, before a data item "x" could be read, a transaction is supposed to obtain a "read-lock" on x. This helps in maintaining a synchronized state of all data residing in a DDBMS.

3. Data Synchronization Concept, approaches and Methodologies

Information have become very important to today's making it a necessary during key decision making. The ability to keep it safe and in perfect condition becomes a priority to ensure that it can be retrieved and that it remains relevant to support decision making when required. Data is normally stored in databases as raw data as collected. The DBMS arranges the data in appropriate record formats which enables easy retrieval, indexing and referral as needed (Ślęzak 2009). It is keen to note that events happens leading to data distortion, DBMS damage as well as loss information. While such information is lost, organizational activities might be crippled or done without a good basing due to lack to inability to access that critical information. Therefore, it becomes critical to have backup strategies which will ensure that information being collected is spread among several storage destinations just to counter this issue of unpredictable inaccessibility of information.

Now we have established how important information is, sharing it have become more vital to firms since it results to improvement of data availability as well as quality. However, during this sharing it is crucial to have all parties using the information to have information that is similar hence the need to have consistency among the sources where the information is stored.

Data synchronization is there for this purpose to ensure that a data set exists in different directory servers (What is Database synchronization? 2019). Such servers need to have data that is consistent and available at all times. Data consistency and availability can be assured by data synchronization and data replication. Data replication involves creation of more than one redundant database copies to improve on data accessibility as well as make the system fault tolerant. Data synchronization in the other hand involves data consistency establishment among several databases with means to ensure that the subsequent updates are done continuously to maintain consistency in all databases. Data synchronization can be challenging but is desirable to organizations. The following are cases where data synchronization might be necessary:

- During data migration
- Information systems regular synchronizations
- Data importation between information systems
- Data set movements in between different environments or stages
- Non-database source data importation

There are several techniques for carrying out data synchronization and companies should choose ways that works for them (Introduction — Data Synchronization: Patterns, Tools, & Techniques 2019) . Depending on the data structures complexity, the process could be termed as either simple or complicated. In most of the time data synchronization is composed of complex related tasks that takes long time to perform them. Some scenarios requires data specialist to redo the whole process of synchronization. Data synchronization process can be difficult and expensive as there are no other standards put in place to optimize it besides

replication. Its maintenance and implementation can be time consuming. Synchronization can be done on systems based on the structure of source and destination databases as shown below:

- Systems where Source and Destination are made of similar structures
- Systems where Source and Destination have dissimilar structures.

In the event where the systems source and destination have similar structures data is compared between various stages and an appropriate data importing is done for consistency (Bacon 1998).

3.1 Source and Destination have dissimilar structures

Synchronization becomes more complicated and synchronization becomes a recurring task. The best scenario is importing data between two software that are maintained by different companies. The process requires running imports on an automatic scheduled basis.

There are four ways of solving data synchronization regardless of the structures similarity level:

- Manually created scripts synchronization
- The data comparison method synchronization which is appropriate where source and target are of similar structure.
- Automatically generated scripts synchronization

3.1.1 Use of Manually Created Scripts

Scripts are manually written to conduct synchronization

Merits

- Open source and free tools can be used
- It is very fast for tables that are indexed
- It is possible to save the script into a stored procedure which makes it possible to run it as an SQL job
- Can at sometimes used as an automatic import

Disadvantages

- Creation of such scripts is tedious since three commands are required for each table i.e. INSERT, UPDATE and DELETE
- Only data that is available as SQL queries can be synchronized this way which eliminates use in data organized as XML and CSV
- Hard to maintain especially when the database structure changes as it calls for modification of three scripts per table.

3.1.2 Data Compare Method

In this method a tool can be used to compare data in both the target and the source. During the comparison process SQL scripts are generated and are used to make sure that the differences between the target and source database are resolved. There a number of programs that are used for data comparison to enable synchronization. They use the same approach where a user is allowed to select both target and source database. In the beginning data is read and then comparison is executed. Additional settings for synchronizing are available in this tools and are necessary for synchronization. They are:

SYNC KEY

The primary key or Unique key constraint is used as the SYNKC KEY. A column combination can be chosen where the primary key does not exist. It used for row pairing between the source and the target rows.

TABLE PAIRING

Tables are by default paired by name. However, this can be changed which allows one to pair on their own needs. For example in some software an SQL query can be used as the source or destination.

SYNCHRONIZATION PROCESS

Once the source and the target are identified the tools proceeds to compare them. Both the source and target data are downloaded and comparison starts based on criteria specified. Values from tables and columns that are equally named are compared by default. Column and table names mapping is supported by all tools. The process optimizes data download. For large volume of data, the checksums are downloaded only. The optimization process is crucial but time requirements for operations performance increases with increase in data volume. During migration an SQL script is generated. This script can run or saved. The script executed can be used to provide an import

Advantages

- Advanced SQL knowledge is not required as it can be done through a GUI
- Visual check ability of differences between databases before synchronization

Disadvantages

- The software used are of commercial use
- Performance decreases with increase in data volume
- Only differences are contained in the generated SQL script. This prohibits from being used for automatic synchronization for future data.

3.1.3 Automatically Generated SQL script synchronization

The method is to that of data comparison only that there is no data comparison involved and the SQL script generated does not contain data differences, but logic of synchronization. The script generated can be saved into a stored procedure which can be run periodically. This method is very helpful for carrying out database automatic imports. This makes the method's performance better than that of compare method.

Advantages

- SQL advanced knowledge is not required.
- GUI setting up is pretty quickly
- SQL script generated can be saved into a stored procedure
- Ability to use the SQL script as an automatic import.

Disadvantages

- The tools are commercialized
- Inability to manually check the differences as the whole process is executed in a single step

3.2 Effective database synchronization strategies, their challenges and ways to resolve such Issues

Whenever coming up with a strategy to use for data synchronization it is important to consider and focus on basic concepts and principles involved in databases (Strategies for Effective Database Synchronization 2019):

- Data consistency
- Partition Availability
- Performance

3.2.1 Data consistency

The traditional relational database can only achieve high consistency levels between the master and slave database by use of shared storage (Scott 2013). This can only be done through Write-Ahead-Logging a technology where whenever an update occurs it updates the operations write logs or also commits the transaction. The method to achieve the consistency between the slave and the master databases is discussed:

When a transaction is committed, a two log writing operation is initiated: The first operation writes the log to the ephemeral disk while the other synchronizes the log to the slave database where it ensures the data is saved on the disk.

The master database returns an acknowledgement once the two operations are successful. At this point the commitment of the transaction is deemed successful.

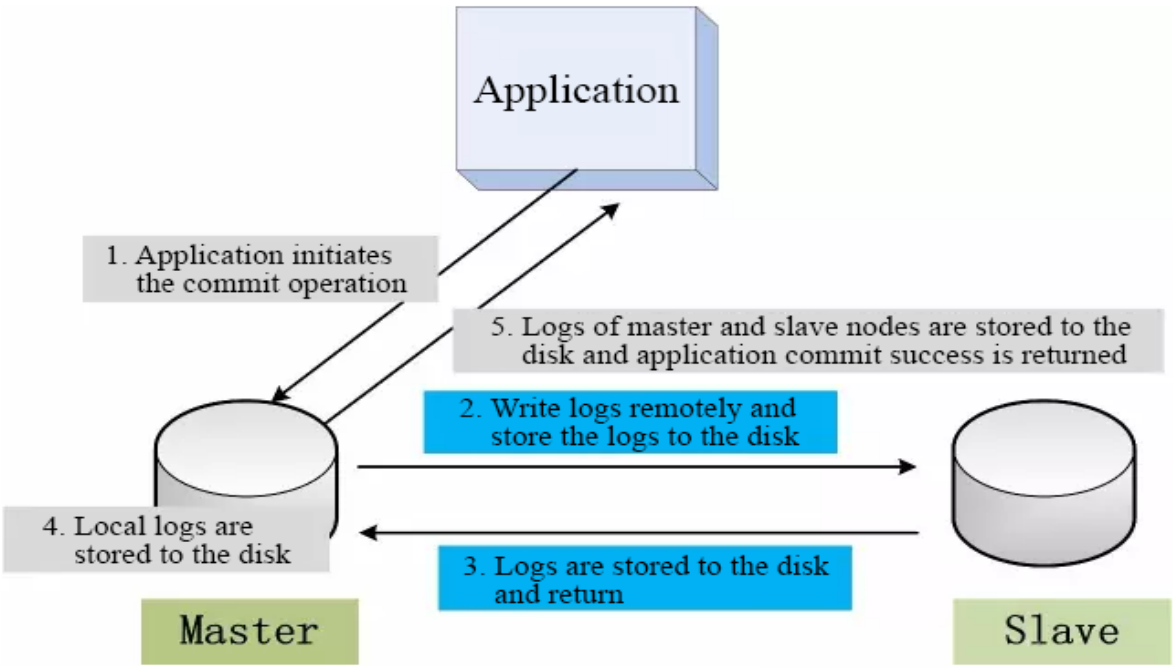

Figure 10:Master-Slave database concept

From the figure above, generated logs by transactions exists in both the slave and master database. A stable synchronization is achieved whenever the committing operation of a transaction is returned to the application.

In the event of a crash by the master database at a particular time, the slave is able to give services since it is consistent with the master database (Loshin [no date]). Transaction data loss is not experienced since strong consistency between the slave and master database is achieved. From this point the issue of data availability is considered next now that strong data synchronization have been obtained. The main purpose of the slave database is to be on standby and take over requests that are master database bound when it fails. The slave database gets promoted to the master database status and provide services. The challenge that occurs in this process is what will take charge of the master-slave switch. This have been done previously through a manual switch but the process is too inefficient.

An automated mechanism has been put in place to avoid the manual nightmare and is done through **HA** (High Availability). It is a testing tool suited to eliminate errors. Its deployment requires introduction of a third server which is connected to both master and slave databases. Whenever the server detects connection failures bound for the master database it switches the provider to the slave database.

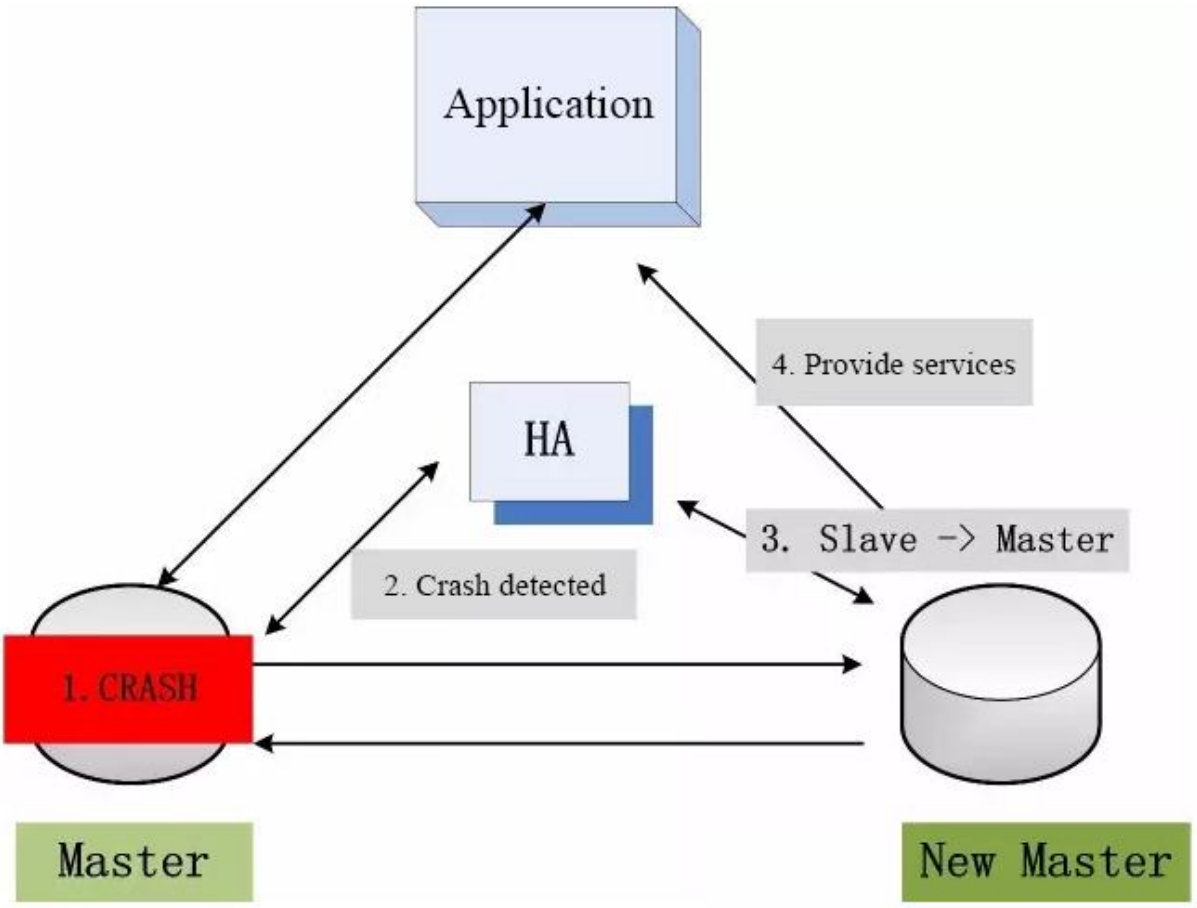

Figure 11: Master database crash

This setup is deemed to provide a perfection solution. However, like any other setup problems may arise making it difficult to operate. Consider the following scenarios:

The database becomes a single point when the master database fails and the slave database takes over. This situation will continue until the moment the master database restarts and comes back for service. Now if

the slave database happens to crash , the system becomes unavailable. This can be solved by having more slave databases and replicas to the system.

Another issues occurs where a network issue arises compromising the connection between the master and the slave database. In this scenario the master database continues to provide services but synchronization stops taking place. In case the master database stops working the system becomes unavailable. Network problem can also occur between the HA and the slave database or the master database as well. Consider the following:

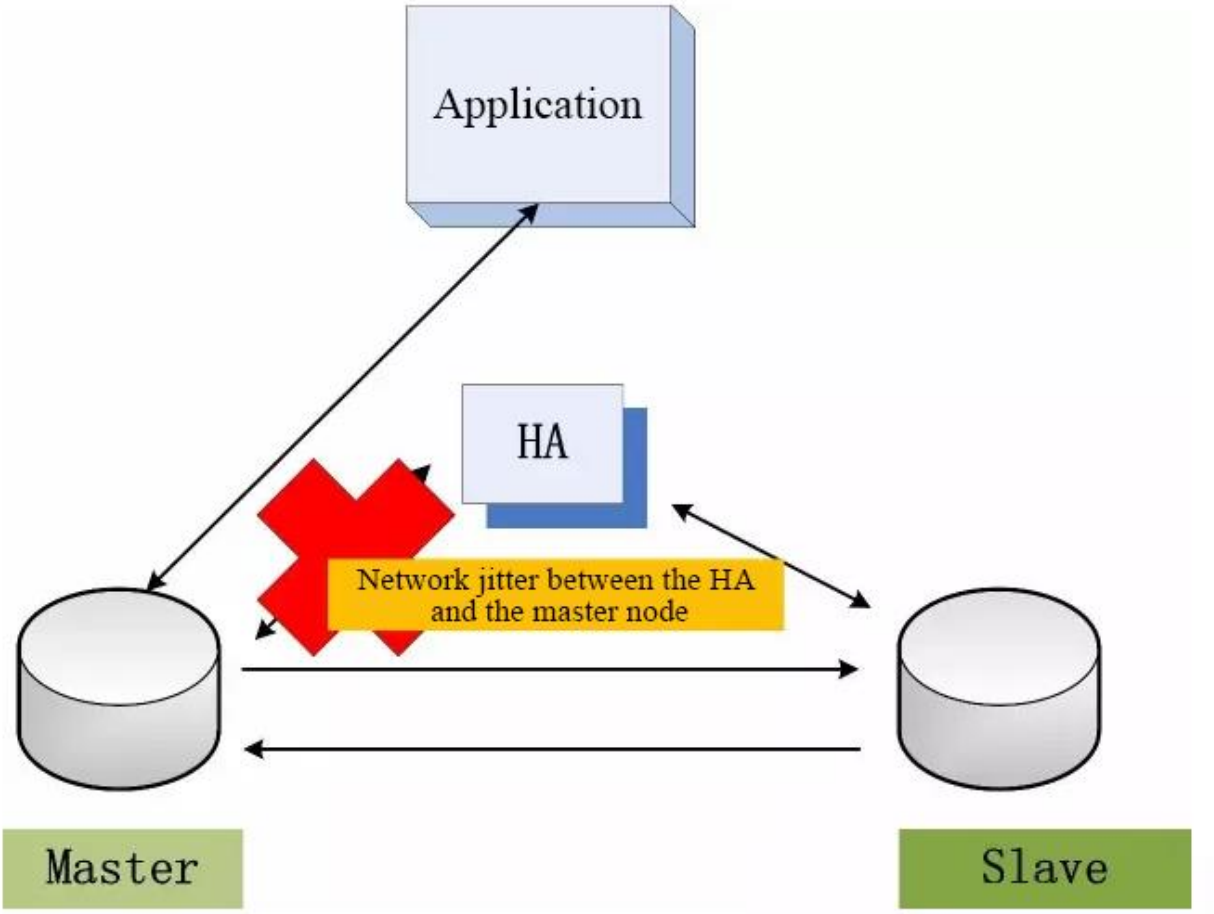

Figure 12:Network Jitters inhibiting communication]

Two potential problems are likely to arise from the above:
The HA fails to find connection to the master and deems it crashed. By default it promotes the slave database to handle requests meant to be done by the master database. The master database however remains on and functioning normally and continues to provide external services. The slave database also continues to provide the same services. As a result dual writing takes place.

The HA fails to find connection to the master and deems it as a network problem. As a result the HA deems the master database operational and refrains from taking any action. However, the master database crashes in the process and the HA again fails to perform any action due to impaired network. In this scenario dual writes are avoided but system availability dented.

The HA itself might crash and stop being operational at the same time. This hampers the master-slave database switching process. Adding an additional HA layer for the original HA is a possible solution but

comes at a cost of efficiency reduction. This results to consistency issues brought about by a distributed environment. Protocols such as Raft and Paxos have been built to deal with such scenarios.

3.2.2 Partition Availability

So above we have looked at how a distributed environment can bring consistency issues as well as effect on overall performance with introduction of more channels. The affected aspect is that of availability which is solved through the introduction of Raft and Paxos protocols. They use system of strong consistency made up of a master and slave databases, HA monitoring and a switching technique between the slave and master databases.

The diagram below illustrates this example

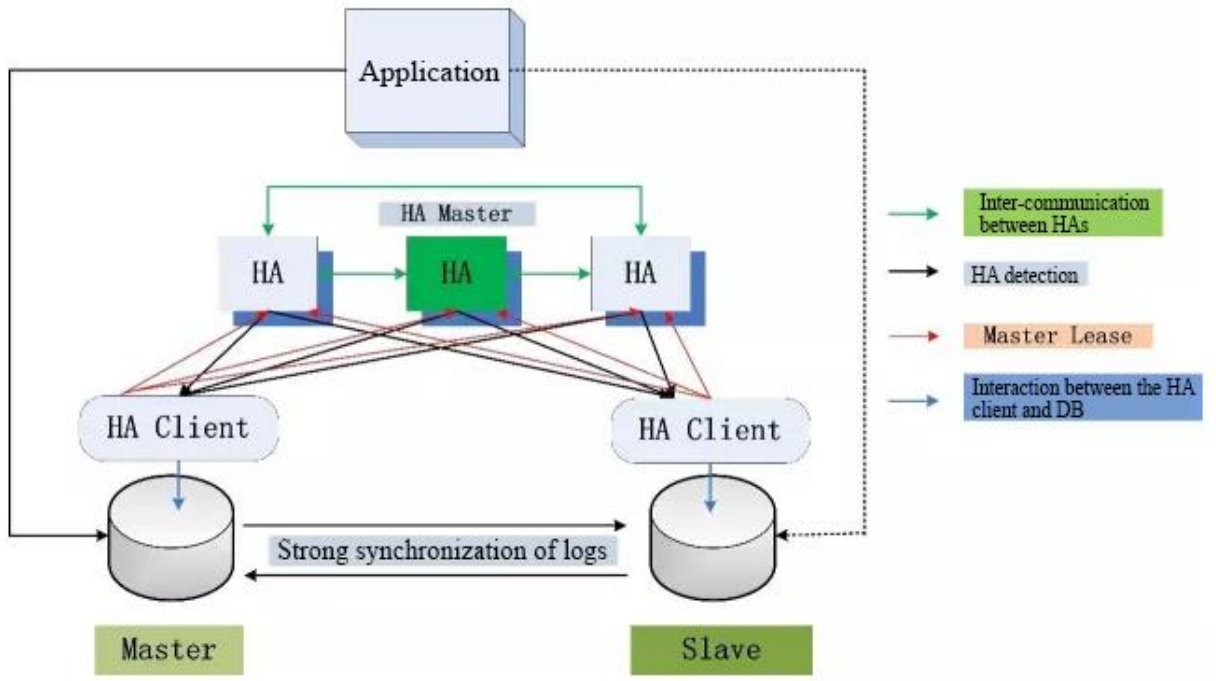

Figure 13: Hybrid master - slave database structure

- The system is a bit complicated. The one-master and one slave mechanism is adopted ensuring strong data synchronization. The following changes are also adopted: HA clients deployment on both slave and master database.
- HA hosts number increment. The system is composed of three HA hosts with one acting as the HA master while others acts as HA participants.
- Two way communication establishment between the HA clients and HA host. The HA host detects if the database location of the HA client can provide services. A communication channel is leased between the HA host master and HA client.
  This structure solves the following problems in the following ways:
  The structure solves the HA software availability issue: This is done by increasing the HA hosts from one to three hosts. The hosts does an election to elect their master automatically either the Raft or Paxos protocol. The three HA hosts deployment and the Raft/Paxos protocol ensures high availability of HA service thus ensuring the HA software's availability.
  The next problem is that of the HA's ability to identify whether in access to the master database is caused by network failure or database crashing. The goal is to ensure that there is always a master database in charge at any circumstances providing external services. The problems

is sorted through a lease mechanism where HA clients are deployed. The master database status is not held permanently and is held for a limited range if time. The master database status is renewed every say 10 seconds where a new lease is initiated by the HA client. As long as a continuing lease status is maintained from the HA master after every 10 seconds, the current master database is not downgraded to slave database. This way the following scenarios are taken care of:

First Scenario: **The master database results into a crash, but the master database's location server runs normally and the HA client runs normally.**

Here the HA client runs normally while the master database crashes. The HA client in turn sends a request in a bid to give the lease of the master database to the HA master. When this is done the HA master proceeds to promote the slave database to be the master database. The initial master database proceeds as the slave database after a rebooting process.

Second Scenario: The master database's host location crashes.

In this scenario both the HA client and the master database crashes at the same time. This results to communication to be cut from the HA master to the HA client. At this instance the HA master is unable to promote the slave database to be the master database as it is unable to know the cause of the issue. The HA master waits for the expiration of the lease time. If it fails to receive a lease renewal request before term expiration period, it automatically proceeds to promote the slave database to become the master database. The initial master database in turn becomes the slave database after a reboot.

Third Scenario: The master database remains normal but the network connection between the master database and the HA master fails.

The HA master in this scenario is unable to differentiate between the second scenario and this scenario and therefore proceeds to process it with same logic as the second scenario. Once the lease term expires and fails to receive any renewal request it promotes the slave database to master database. But before this occurs the HA client of the previous master database needs to do some functions. Since the HA client of the initial master database fails to receive a response on requests concerned with lease renewal request from the HA master due to network issue and the lease term has expired, it automatically downgrades the master database to slave database. By doing this the dual master databases issue is eliminated now dual writes occurs.

At this point it can be concluded that continuous availability of the database system together with a robust consistency of the master is very much possible. The renovation of the existing database is also feasible but with actual implementations the process is highly sophisticated. The implementation of this master-slave system within the database. Through the HA client the database system is able to downgrade between the slave and the master databases.

3.2.3 Performance

Consider the illustration below.

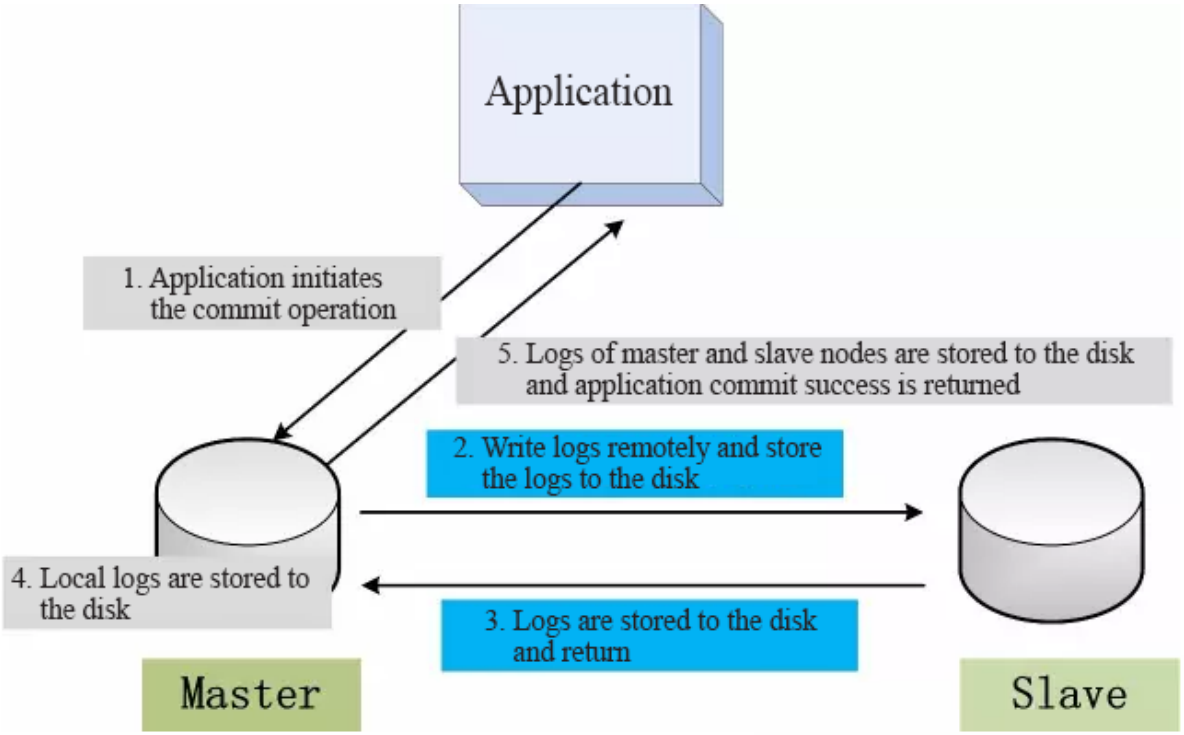

Figure 14:master-slave database

The basic master-slave system is created to ensure consistency through data synchronization. Once a commit transaction request is initiated, it has to be synchronized with the transaction logs of the slave database. This ensures a robust data synchronization where the logs are saved in the disk in the process. The synchronization process is nearly instant since the network interaction between the slave and the master databases is perfected.

The main question that remains is on the performance of the system due to addition of all these structures. The latency time between the slave and master database together with the slave database disk performance comes into play during performance evaluation.

Deployment of multiple slave databases is one of the ways to improve performance. In this way as long as a single slave database has completed log synchronization and a response have been returned to the master database and the logs stored to the disk, the commit return operation is considered successful. This deployment of multiple slave databases becomes helpful in elimination of network errors.
The figure below shows a way through which performance is optimized.

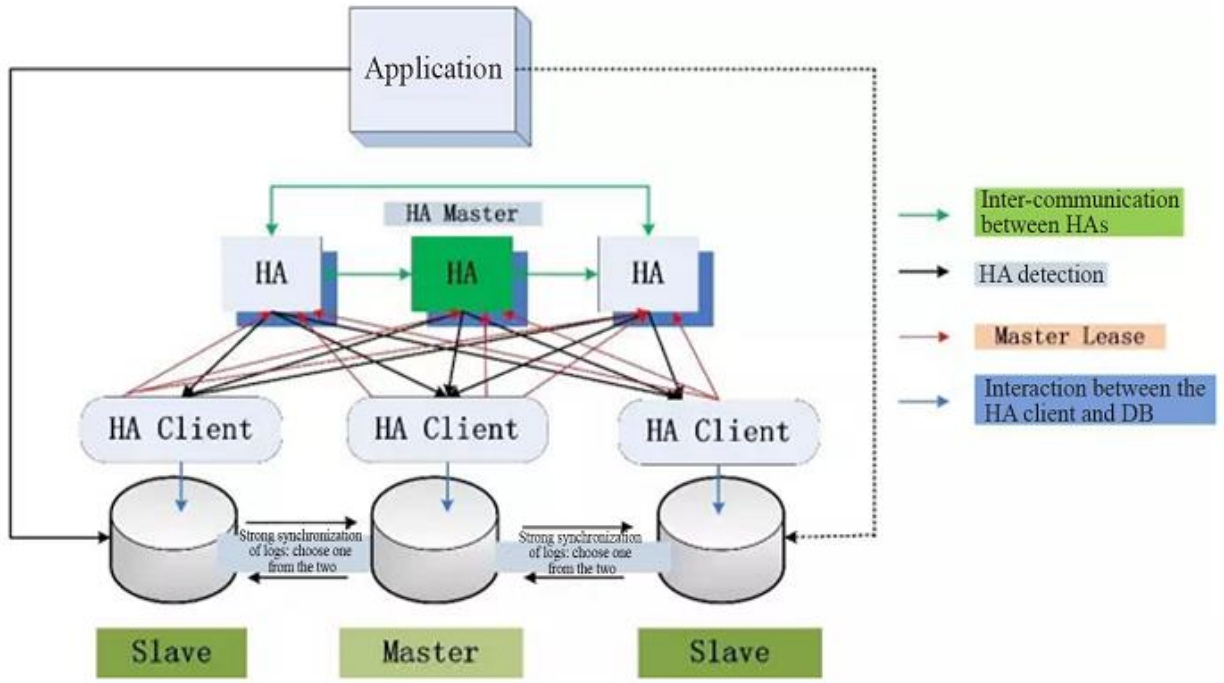

Figure 15: Two slave database structure in use

From the diagram above a third database is introduced hence three data copies are generated. A transaction is committed once log synchronization is completed on one of the two slave databases which reduces the impact of network issues usually experienced in one slave database. The extra copy can be used to solve the security issues related to data once the master database crashes. A single point of failure is avoided in the event the master database crashes where the two data copies are in a position to provide services.

Another problem is introduced in the event the master database where there is an issue on determining which slave database takes over as the master database. The following criteria can be used:

Log first: The HA master elects the slave database that has the recent logs as of the master database

Priority based on host: If both the slave databases has the recent logs a priority based selection is used using their IP addresses where the one with the lower IP address is selected.

Once a new master database is elected log synchronization is done with the remaining slave database. The new master database proceeds to provide service to applications.

4. METHODOLOGY APPROACH ADOPTED

4.1 Synchronization Technique

This section describes the algorithm that is used to synchronize changes. Figure 16 below shows the high-level architecture diagram of the database synchronization process.

The client-side database is synchronized with the server database using a client-side application hosted on the web server. This communicates about any connection establishment and data transfer with the server-side database in PostgreSQL. The detailed approach used is further described in this section.

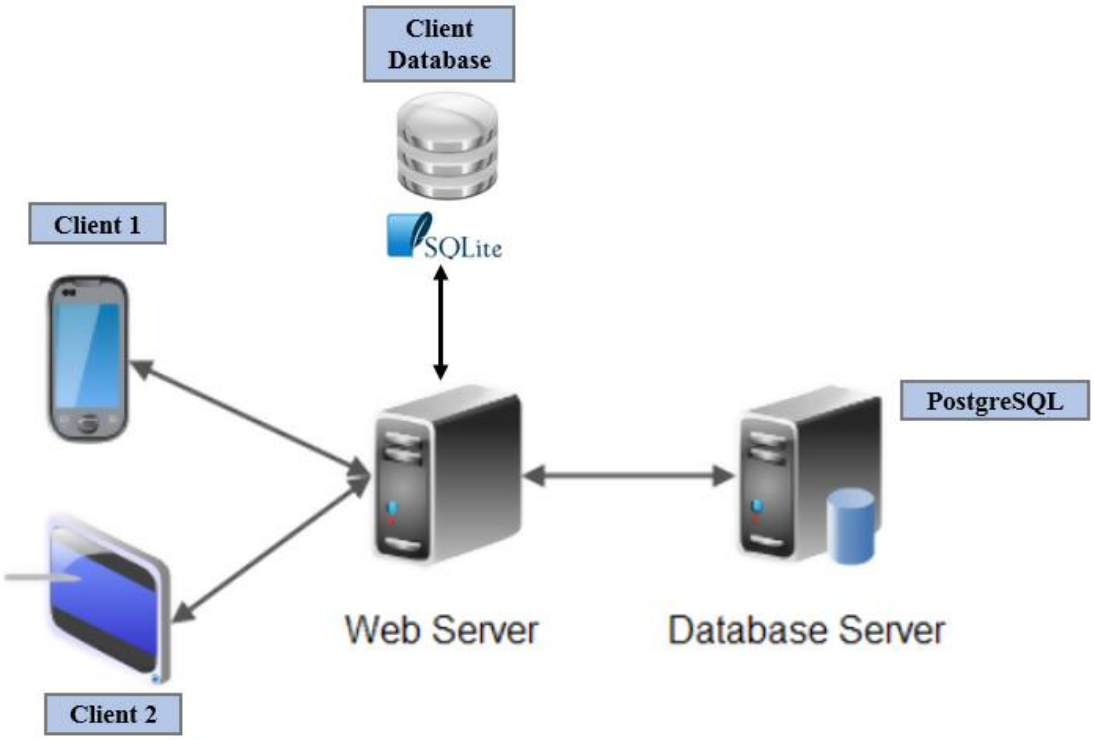

Figure 16:Architecture diagram of database synchronization

This section describes the algorithm that is used to synchronize changes. There are five approaches available to synchronize the data. Table 1 [2] below compares the various methods and lists their advantages and disadvantages.

	Bandwidth efficiency	Storage efficiency	Track server changes	Track local changes	Computational complexity
Timestamp synchronization	GOOD	GOOD	GOOD	AVERAGE	GOOD

Status flag synchronization	GOOD	GOOD	POOR	GOOD	GOOD
Wholesale synchronization	POOR	GOOD	N.A.	N.A.	GOOD
Mathematical synchronization	AVERAGE	GOOD	N.A.	N.A.	POOR
Log synchronization	GOOD	Poor	GOOD	GOOD	GOOD

Table 1. Advantages and disadvantages of different synchronization models

The synchronization algorithm keeps the local database and remote server synchronized. The synchronization is initialized by the client. When a request is made the algorithm executes the following steps:

1. Uploading changes to Server.
2. Solving Conflicts on server.
3. Downloading and applying changes on local database that is returned from server.

At first, a temporary table called *change_history* table is maintained where all the changes that are collected are converted to xml file and made ready for uploading these changes. These xml tags are present in *Change* column of *change_history* table as shown in figure 17 below.

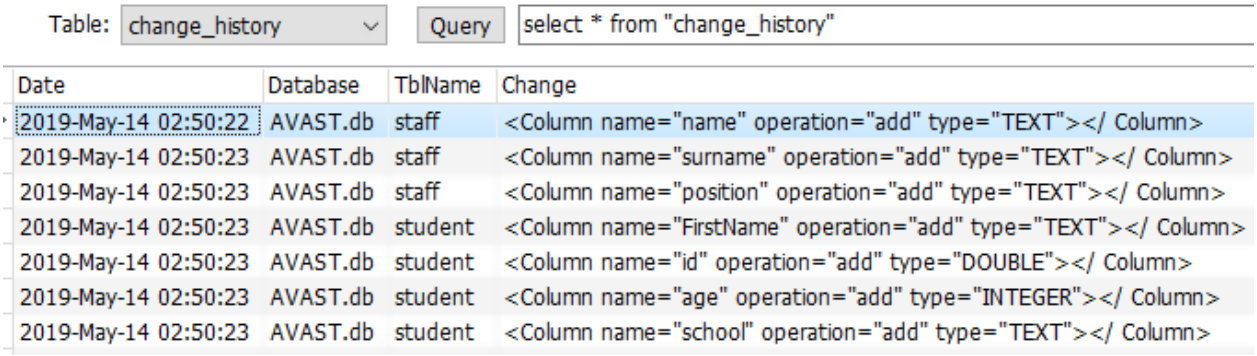
Table: change_history Query select * from "change_history"

Date	Database	TblName	Change
2019-May-14 02:50:22	AVAST.db	staff	<Column name="name" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	staff	<Column name="surname" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	staff	<Column name="position" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	student	<Column name="FirstName" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	student	<Column name="id" operation="add" type="DOUBLE"></ Column>
2019-May-14 02:50:23	AVAST.db	student	<Column name="age" operation="add" type="INTEGER"></ Column>
2019-May-14 02:50:23	AVAST.db	student	<Column name="school" operation="add" type="TEXT"></ Column>

Figure 17: change_history table with Change column

All changes in *change_history* table is a part of synchronization process and nothing is restricted in these changes. Next, the *change_history* table is cleared when the xml is uploaded as there is no need for this anymore.

The main challenge is solving conflict resolution which is handled with the help of mandatory flag. As a final step to keep them synchronized, remote changes are applied as well. Since there are multiple clients getting involved, the changes on server cannot be deleted because there could be still clients which are still offline and need to be synchronized. In this case, the timestamp will be needed for clients to know which changes occurred on server after the last synchronization. Therefore, it is essential for clients to store the timestamp they had last synchronization at. For the next synchronization, it looks for only changes in remote server with timestamps which are greater than the timestamp on the client file. At the server-side, the timestamps are incremented by 1 after each synchronization.

4.2 Resolving data conflict between client and server

During data synchronization between client and server (the central database), the most critical part to be addressed is providing a resolution to synchronizing conflicting changes. It is very crucial to handle data conflict because it can lead to data loss, decreased data integrity and corrupt data. If two different changes are made in two different datasets, then the changes can be applied directly without having the possibility of data lost. But, if the changes are made in the same dataset, it is very likely to lose data. Consequently, the changes will be applied which will overwrite one of the datasets.

To resolve this problem, consider the following possible scenario for data conflict.

1. Client retrieves the data from the server and then the connection with the server is closed.
2. The client then changes the specific row and column. Let us assume row x and column y to value *VAL1*.
3. Meanwhile, the server also makes a change on row x and column y to value *VAL2*.
4. Now the connection is online, and the system attempts to synchronize the two datasets.
5. The system faces obvious data conflict during synchronization.

The question is, which one (VAL1 or VAL2) will be accepted in row X and column Y? Every database synchronization tool has its own strategy to solve conflict resolution.

The main strategies are the followings:

1. Recent Data Wins. Take the data item according to last updated time
2. Client Wins. Take the data item of the client.
3. Server Wins. Take the data item of the server.

Our strategy in DBSync is different for data resolution from other frameworks. If a row is marked as mandatory (locked) on server side - it will win. If it is marked as non-mandatory (unlocked) - client will win. For example, each configuration value in the database will contain data value as well the mandatory flag (mandatory/monitored) which handles the sync direction in case of data conflict.

Mandatory flag is part of the data and is changed in server:
If mandatory flag is locked (mandatory flag = 1)
    it is synchronized from server to client (Server wins)
If mandatory flag is unlocked (mandatory flag = 0)
    it is synchronized from client to server (Client wins).

The main challenge is resolving the data conflict which is handled with the help of mandatory flag. As a final step to keep them synchronized, remote changes are applied as well. Since there are multiple clients getting involved, the changes on server cannot be deleted because there could still be clients present, which are offline and need to be synchronized. In that case, the last updated timestamp for clients will be read, to know which changes occurred on server after the last synchronization. Therefore, it is essential for clients to store the timestamp they had their last synchronization at. For the subsequent synchronization, the system looks for only changes in remote server with timestamps which are greater than the timestamp on

the client file. As mentioned previously, the timestamps at the server-side are incremented by 1 after each synchronization.

4.3 Data Representation of changes

change_history table has four columns which are the attributes for change:

1. *Date* – Indicates the exact time of change occurrence. The format is DATE TIME (yyyy-mmm-dd hh:mm:ss.s)
2. *Database* – Indicates the name of database where change occurred
3. *TblName* – Indicates the name of table where change occurred
4. *Change* –XML Tag and providing all required information related to change

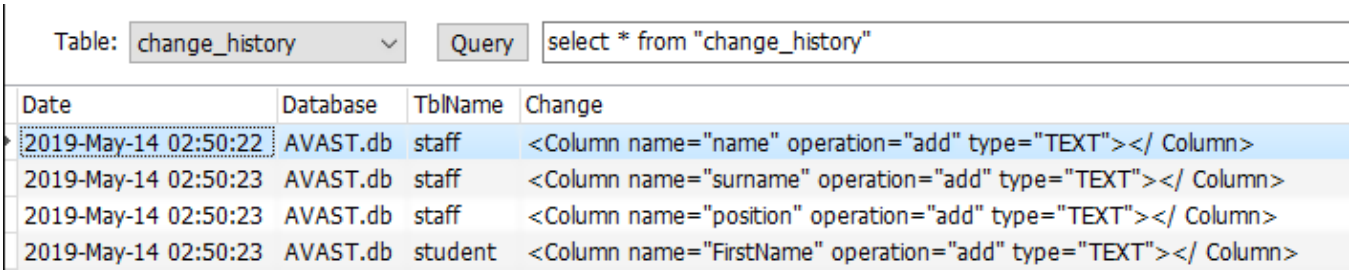
Table: change_history Query select * from "change_history"

Date	Database	TblName	Change
2019-May-14 02:50:22	AVAST.db	staff	<Column name="name" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	staff	<Column name="surname" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	staff	<Column name="position" operation="add" type="TEXT"></ Column>
2019-May-14 02:50:23	AVAST.db	student	<Column name="FirstName" operation="add" type="TEXT"></ Column>

Figure 18:Sample data representation of change_history table

Figure 18 shows the sample data representation of the *change_history* table. Within this xml tag it is possible to characterize every change. The columns that are related to the changes are defined inside *Column* element with its attributes such as operation type, value type, name etc. Row related edits are defined in *Row* element which has the same attributes as in *Column*. There are 2 types of table operations - remove and clear.

Table related changes are defined within *Table* element. The following are the database operations and corresponding xml tag which defines what change appeared after this operation:

Adding column. Column name and data type must be specified.

<Column name="" type="" operation="**add**"/>

Removing column. Column name and data type must be specified.

<Column name="" operation="**remove**" />

Clear table.
<Table operation="**clear**"/>

Remove Table.

<Table operation="**remove**"/>

Deleting row. Row index must be specified.
<Row operation="**remove**" index=""/>

4.4 Technical Specifications

4.4.1 Platform

The research thesis is carried out in the during the internship at Avast Software.

1. Operating system: Windows Server 2016 GPU machine
2. Client database – SQLite version 3.28.0
3. Server database – PostgreSQL version 11
4. Client-side synchronization tool development - C++

4.5 Log based data synchronization technique

The log based synchronization technique has become a reliable way through which data synchronization is conducted. Logs contains all records of database operations carried out by applications that accesses particular databases. These logs are records of all transactions that occurs from applications accessing the database. Once an operation is carried out on the database it is committed to mark the transaction complete resulting to a database update. The study adopts this method to carry out research on data synchronization.

As it was discussed in figure above the client-side database is synchronized with the server database using a client-side application hosted on the web server. This communicates about any connection establishment and data transfer with the server-side database in PostgreSQL. The synchronization tool is developed in C++.

In the implementation tool there exists three main classes that class CReport, DbsFormatConvertor and CReportSync.

The class CReport defines all database operations and is responsible for structure creations, their deletion as well as insertions, updating and same to deletion. Advanced SQL Statements are used in CReport class methods. CReportSync class inherits CReport class. This class is called whenever an application wants to make some changes on the database and store all that changes in “change_history” table. The saved logs contain information on transactions carried out. The logs are written on the client side database through the instructions defined above. CReportSync overrides the same base class methods which are responsible for table operations . Whenever this methods are called AddChange method will get called for saving logs. Once logs are successfully saved, related base class function will get called and will return exit code.

```
class CReportSync : public CReport
{
    CReport m_dbsHistory;

    static constexpr auto           DATE     { "Date" };
    static constexpr auto           DATABASE { "Database" };
    static constexpr auto           Table    { "TblName" };
    static constexpr auto           CHANGE   { "Change" };
    static constexpr auto           timeFmt  { "%Y-%b-%d %H:%M:%S" };

    bool lockChange = false;

public:
    CReportSync();

    ~CReportSync();

    ErrCode                         SetDouble(uint32_t rowIdx, uint32_t columnIdx, double d) override;
    ErrCode                         SetDouble(uint32_t rowIdx, const std::string & columnName, double d) override;
    ErrCode                         SetInt(uint32_t rowIdx, uint32_t columnIdx, int64_t i) override;
    ErrCode                         SetInt(uint32_t rowIdx, const std::string & columnName, int64_t i) override;
    ErrCode                         SetText(uint32_t rowIdx, uint32_t columnIdx, const char * text) override;
    ErrCode                         SetText(uint32_t rowIdx, uint32_t columnIdx, const wchar_t * text) override;
    ErrCode                         SetText(uint32_t rowIdx, const std::string & columnName, const char * text) override;
    ErrCode                         SetText(uint32_t rowIdx, const std::string & columnName, const wchar_t * text) override;
    ErrCode                         ReadTable(const std::string & tableName) override;

    ErrCode                         ClearTable();

    bool                            AddColumn(const std::string & columnName, const CSQLite::DataType type) override;
    bool                            DropColumns(const std::set<uint32_t> & columnIdxs) override;
    bool                            DropColumnsByName(const std::set<std::string> & columnNames) override;
    bool                            DropTable(bool clear = true) override;
    uint32_t                        CreateNewRow(void) override;
    bool                            DeleteRow(uint32_t rowIdx);

    void                            AddChange(const char * change);
};
```

Figure 19:report_sync.h

4.5.1 Use of a middleware

Pugixml which is a light-weight C++ XML processing library. Pugixml enables very fast, convenient and memory-efficient XML document processing. **CreateDbsFromXml** method is able to create a database from the XML file. This method is used to create database on client side, define a table structure and other initialization purposes. The data are parsed by means of Pugixml library and database operations carried by CReport class whereas the **CreateXmlFromDbs** creates XML from a database. **CreateXmlFromChange** is resposible for creating xml from changes that are logged during database operations.

Through the use of these three functions, the databases are able to coordinate to communicate and therefore enhance synchronization of data between themselves.

```
#include <pugixml/pugixml.hpp>

class DbsFormatConvertor
{
    CSQLite::DataType StrToDataType(const std::string & dataType);
    const char * DataTypeToStr(const CSQLite::DataType dataType);
    ErrCode Update(const pugi::xml_node & node, CReport & report);
    ErrCode MergeChange(std::string, pugi::xml_node & node);

public:
    DbsFormatConvertor();
    ErrCode CreateDbsFromXml(const TCHAR * filename);
    ErrCode CreateXmlFromDbs(const std::string & dbsName);
    ErrCode CreateXmlFromChanges();
    ErrCode ConflictResolution(std::string & clientChanges, std::string & serverChanges,
        std::map<std::string, bool> columnFlags,
        std::map<int, bool> clientRowFlags,
        std::map<int, bool> serverRowFlags);
    ErrCode Synchronize(std::string & dbsName, std::string tblName, pugi::xml_node & update);
    ~DbsFormatConvertor();
};
```

Figure 20:synchronization enabling function

4.6 Unit Tests

We will use unit tests to verify that our implementation of DbSync works as planned and that the quality of our code is high. These unit tests mainly serve to verify that the software modules, classes work properly and to detect regression when there is some update on code. They test if the individual components handle the different synchronization situations correctly, including conflicts.

4.7 Other Tests

Other tests especially, Bandwidth Consumption Tests can not applied on current stage since the server part has not been implemented.

5. CONCLUSION

The thesis research on the study and a working tool for database synchronization with conflict resolution is based on experiments on real data to show how the synchronization techniques can solve the data inconsistencies and guarantee the integrity of the data. It offers and deep insight on the concepts of data synchronization software development and its application to various kinds of client-side databases and discusses its challenges and future directions in the field of database synchronization. Being familiar with the concepts and advantages, as well as limitations of DB Synchronization was essential in leveraging its potential in my research with the goal of developing strong knowledge-base and skill-set in database programming.

The goal of this study was to study how data synchronization solves inconsistencies issues as well as maintain the integrity of data at all times. The study adopted a log base synchronization technique where the logs were used to in the process of ensuring that both the client and server databases were synchronized through the transaction logs to ensure that both were consistent. The study looked at how conflict resolution is administered offering solutions to challenges that arises with synchronization. Therefore, the study satisfied those goals set at the beginning of the study.

Future study should include a fall back method which introduces the aspect of a slave database and High Availability in this study that monitors the availability of both the server and the client database to ensure that there is always a database in place to offer services to applications.

REFERENCES

1. BACON, JEAN, 1998, Concurrent systems. Harlow, England : Addison-Wesley.
2. CARUCCIO, LOREDANA, POLESE, GIUSEPPE and TORTORA, GENOVEFFA, 2016, Synchronization of Queries and Views Upon Schema Evolutions. ACM Transactions on Database Systems. 2016. Vol. 41, no. 2, p. 1-41. DOI 10.1145/2903726. Association for Computing Machinery (ACM)
3. How Does One-Way vs. Two-Way Data Synchronization Work? - skysync, 2019. skysync [online],--- (How Does One-Way vs. Two-Way Data Synchronization Work?
4. Introduction — Data Synchronization: Patterns, Tools, & Techniques, 2019. Datasyncbook.com [online].
5. LOSHIN, DAVID, [no date], Master data management.
6. SCHLAGETER, GUNTER, 1978, Process synchronization in database systems. ACM Transactions on Database Systems. 1978. Vol. 3, no. 3, p. 248-271. DOI 10.1145/320263.320279. Association for Computing Machinery (ACM)
7. SCOTT, MICHAEL LEE, 2013, Shared-memory synchronization. San Rafael, Calif. : Morgan & Claypool.
8. ŚLĘZAK, DOMINIK, 2009, Database theory and application. Berlin : Springer.
9. Strategies for Effective Database Synchronization, 2019. Alibaba Cloud Community [online].
10. WELLS, APRIL J, 2005, Grid database design. Boca Raton, FL : Auerbach Publications.
11. What is Database synchronization?, 2019. DBConvert [online].
12. ZIVKOVIC, MARKO and &RARR;, VIEW, 2019, What is SQL Data Sync. SQL Shack - articles about database auditing, server performance, data recovery, and more [online]. 2019. [Accessed 6 July 2019]. Available from: https://www.sqlshack.com/what-is-sql-data-sync.
13. ANAND, S., GOPALANI, N., KIM, G., HUNT, N., & RAU, S. R. 2012. *U.S. Patent No. 8,315,977*. Washington, DC: U.S. Patent and Trademark Office.
14. MALHOTRA, N., and CHAUDHARY, A. 2014, Implementation of Database Synchronization Technique between Client and Server. *International Journal of Engineering Science and Innovative Technology*.
15. MEDENICA, Z., & KUN, A. L. 2012, Data synchronization for cognitive load estimation in driving simulator-based experiments. In *Adjunct Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI'12)* (pp. 92-94).
16. ÖZSU, M. T., and VALDURIEZ, P. 2011, *Principles of distributed database systems*.
17. Springer Science & Business Media.

Acronyms

XML – Extensible Markup Language

DDBMS – Database Management System

Contents of enclosed CD