

**UNIVERSITY OF NAIROBI**



**SCHOOL OF COMPUTING AND INFORMATICS  
MASTERS OF SCIENCE IN COMPUTER SCIENCE**

**DYNAMIC LOAD BALANCING MIDDLE-WARE FOR HETEROGENOUS DATABASE  
PARTITIONS**

**CASE STUDY: MYSQL DATABASES**


**SUBMITTED BY; NDUNGU, K BONIFACE**

**P58/73285/2009**

**09<sup>th</sup>, May, 2012.**

# DECLARATION

This project as presented in this report is my original work and has not been presented for any other University Award.

Signed.....

Date.....12/07/2012

Boniface Kariuki Ndungu.

P58/73285/2009.

The project has been submitted in partial fulfillment of the Requirements for the Degree of Master of Science in Computer Science at the University of Nairobi with my approval as the University supervisor.

Signed.....

Date.....13/07/12

Mr. Tonny K. Omwansa.  
(Supervisor)

## ***Abstract***

In the recent past we have seen people and organizations appreciate the use of Information and Communication Technology (ICT), as this happens massive data has been generated, replicated and stored for future use. These dataset has grown to unimaginable size the I.T experts 10 years ago couldn't have predicted. We have seen institutions like bank and telecommunications industry acquire millions of customers whose detailed information need to be stored in database system and retrieved and manipulated at will. In the banks for example the customers will want to access their account any time of the day regardless of whether there is no power or not, or if there are so many customers that the customer connected to the servers and each have to wait for their turn in the queue.

Due to these demands computer scientist and expert from various industries have put their heads together to try and find a solution to make sure that as the information dataset continues to grow the daily operation of any computer system continues to give the same expected output or better. Hence these experts have seen the need to continually create the best strategies to ensure total availability of information in an information system. One of these efforts has been noticed in the area of distributed database management systems which has initiated the need to have mechanisms like load-balancing to ensure high availability, fail-over, increased response and much more. This has lead to a more efficient way of making sure that life move smoothly for those people using information and communication technology services.

## ***Acknowledgement***

First and foremost I thank the almighty God for giving me knowledge and wisdom to tackle my day to day life and also for the academic knowledge he has bestowed on me. Likewise, I would like to acknowledge all those people who supported me during my project, especially the members of panel four who guided me personally on this project. I would also wish to appreciate my classmate Wycliffe Rono for guidance and support he gave me when writing this report.

I wish everybody who contributed to this project directly or indirect best of luck in their lives an may God grant them more for their good spirit.

Thanks a lot.

## Table of Contents

Chapter 1: Introduction .....	1
1.1 Background Information .....	1
1.2 Problem Statement .....	2
1.3 Problem Justification.....	3
1.4 Project Objective.....	3
1.5 Research Outcome .....	4
1.6 Scope .....	4
1.7 Limitations .....	4
Chapter 2: Literature Review.....	5
2.1 Overview .....	5
2.2 MySQL Cluster .....	5
2.2.1 NDB Storage Engine.....	6
2.2.2 MySQL Cluster Components.....	7
2.2.3 MySQL Cluster Implementation.....	7
2.2.4 MySQL Cluster Configuration.....	8
2.3 MySQL Proxy .....	9
2.3.1 Proxy Scripting Direct Injection .....	10
2.3.2 Proxy Load Balancing.....	10
2.4 Replication .....	11
2.4.1 Multi-Master Replication.....	12
2.4.2 Master-Slave Replication.....	12
2.5 Load Balancing .....	13
2.5.1 Static Load Balancing .....	13
2.5.2 Dynamically Load Balancing.....	13
2.5.3 Load Balancing Strategies.....	14
Chapter 3: Methodology and Design .....	16
3.1 Project Requirement.....	17
3.1.1 Functional Requirement.....	17
3.1.2 Non Functional Requirement .....	17
3.1.3 Non Functional Requirement .....	18
3.2 System Architecture .....	18
3.2.1 Middle-Ware Architecture.....	18
3.2.2 Process Flow .....	20
3.3 Theoretical Analysis of Load Balancing .....	21
3.4 Description of Load Balancing Algorithm.....	21
3.4.1 Performance Parameters of Server.....	21
3.4.2 Performance Parameters Optimization .....	22
3.4.3 The Load Balancing Algorithm.....	23
3.5 Implementation .....	25
3.5.1 Development environment.....	25
3.5.2 Implementation tools.....	25
3.5.3 Why JAVA and MySQL Databases .....	26
Chapter 4: Results Analysis .....	27

4.1 Discussion of Results .....	27
4.1.1 System Result.....	27
4.1.2 Test Results .....	28
4.2 Challenges Facing Database Querying in Federated Database.....	32
4.3 Evaluation of Available Database Load-balancing Techniques. ....	32
4.3.1 Analysis of different load balancing techniques. ....	34
Chapter 5: Conclusion.....	35
5.1 Achievement.....	35
5.2 Challenges .....	35
5.3 Further Study.....	36
5.4 Conclusion .....	36

<b>FIGURE 1: A BASIC ILLUSTRATION OF MYSQL CLUSTER SETUP.....</b>	<b>7</b>
<b>FIGURE 2: INJECTION OF QUERY IN TO THE QUERY QUEUE.....</b>	<b>10</b>
<b>FIGURE 3: LOAD BALANCING USING MYSQL PROXY.....</b>	<b>11</b>
<b>FIGURE 4: ILLUSTRATION OF A MYSQL DATABASE REPLICATION.....</b>	<b>11</b>
<b>FIGURE 5: CONFIGURATION OF MYSQL MASTER IN MY.CNF FILE.....</b>	<b>12</b>
<b>FIGURE 6: CONFIGURATION OF MYSQL SLAVE IN MY.CNF FILE.....</b>	<b>12</b>
<b>FIGURE 7 : DYNAMICALLY LOAD BALANCING IN A HETEROGENEOUS DATABASE PARTITION.....</b>	<b>14</b>
<b>FIGURE 8: AGILE SOFTWARE DEVELOPMENT METHODOLOGY.....</b>	<b>16</b>
<b>FIGURE 9 : LOAD BALANCING MIDDLE-WARE ARCHITECTURE.....</b>	<b>19</b>
<b>FIGURE 10: LOAD BALANCING MIDDLE-WARE PROCESS FLOW.....</b>	<b>20</b>
<b>FIGURE 11: SERVER LISTENER WAITING FOR A CLIENT CONNECTION.....</b>	<b>27</b>
<b>FIGURE 12: SERVER COMPUTATIONS ON MEMORY USAGE AND CPU USAGE.....</b>	<b>27</b>
<b>FIGURE 13: CLIENT COMPUTATIONS AND RESULTS.....</b>	<b>28</b>
<b>FIGURE 14: GRAPH SHOWING RESULTS FOR EACH IP/SERVER WITHOUT LOAD INTRODUCED.....</b>	<b>29</b>
<b>FIGURE 15: GRAPH SHOWING RESULTS FOR EACH IP/SERVER WITH LOAD INTRODUCED.....</b>	<b>30</b>
<b>FIGURE 16: A GRAPH OF TRIALS AGAINST RESPONSE TIME IN MILLISECOND.....</b>	<b>31</b>
<b>FIGURE 17: MYSQL CLUSTER TEST ENVIRONMENT.....</b>	<b>34</b>

<b>TABLE 1: RESOURCE USAGE FOR ALL THE SERVERS WHEN LOAD IS NOT INTRODUCED. ....</b>	<b>29</b>
<b>TABLE 2: RESOURCE USAGE FOR ALL THE SERVERS WHEN LOAD IS INTRODUCED.....</b>	<b>30</b>
<b>TABLE 3:TRIALS FOR NODES RESPONSE TIME. ....</b>	<b>31</b>
<b>TABLE 4: EVALUATION OF VARIOUS LOAD BALANCING TECHNIQUES.....</b>	<b>34</b>



**DEFINITION OF TERMS:**

**Partition:** A partition is a division of a logical database or its constituting elements into distinct independent parts.

**Heterogeneous Database System** is an automated (or semi-automated) system for the integration of heterogeneous, disparate database management systems to present a user with a single, unified query interface.

**Replication-** is the frequent electronic copying data from a database in one computer or server to a database in another so that all users share the same level of information.

**Load Balancing-** Distributing processing and communications activity evenly across a computer network so that no single device is overwhelmed.

**Middle-ware-**is a general term for any programming that serves to "glue together" or mediate between two separate and often already existing programs.

**CPU-**Central Processing Unit.

**TCP-**Transmission control protocol.

**UDP-**User datagram protocol.

**SCTP-**Stream control transmission protocol.

**MySQL-**It's an open source relational database management system.

# Chapter 1: Introduction

## 1.1 Background Information

Sharing information among autonomous heterogeneous databases has been researched extensively. In essence the problem has been to make component databases inter operable despite their different platforms (software and hardware) [3]. In many large organizations there has been a proliferation of database systems to handle ever increasing volumes of information. These systems tend to be developed in isolation, and this result in structural and semantic heterogeneity, and related problems. The promise of a commercial competitive edge via the logical integration of existing database systems, has attracted intense interest. A major assumption has been that component databases have a-prior knowledge of remote schema. However, this is only reasonable provided the number of participating databases (and global information) is small. Recent advances in communications technology have led to expectations of large scale, world wide database interoperability. There are various fundamental difficulties associated with large scale database interoperability. These include scale, autonomy and heterogeneity.

A Heterogeneous Database System is an automated (or semi-automated) system for the integration of heterogeneous, disparate database management system to present a user with a single, unified query interface. Heterogeneous database systems (HDBS) are computational models and software implementations that provide heterogeneous database integration [1,2]

Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The load balancing service is usually provided by dedicated software or hardware, such as a multilayer switch or a Domain Name System server [4].

MySQL is a widely used open source database. When MySQL is used in a cluster, one node is selected

as master. The master's job is to distribute changes to the data to the other MySQL nodes. These nodes are named slaves. Changing data on a slave will not distribute the changes to the rest of the cluster. Therefore, the slaves are only capable of handling read-only queries. Another problem with the clustering capabilities in MySQL is that it provides no means to distribute the transaction load. Distributing load is usually handled by a load-balancer. A load-balancer selects on which node a request is to be executed. The goal for the load-balancer is to equalize the load on all nodes, to avoid bottlenecks and to achieve maximum throughput. Some load-balancers blindly follow a specific pattern to reach this goal, others evaluate information obtained from the nodes and base their decision on that.

## **1.2 Problem Statement**

Large volume of data mostly poses the most serious problem for which many organizations have data warehouses. The amount CPU execution time and resources needed to query data across the networks is enormous. Key measure of performance for a computing system is speed, response time or execution time or latency and throughput. Reducing execution time will nearly always improve throughput; the reverse is not true.

Execution time can mean:

- Elapsed time -- includes all I/O, OS and time spent on other jobs
- CPU time -- time spent by processor on your job (no I/O)
- CPU time can mean user CPU time or System CPU time

Example:

*Job1 = Total time to complete 250 ms (quantum 100 ms).*

1. First allocation = 100 ms.
2. Second allocation = 100 ms.
3. Third allocation = 100 ms but *job1* self-terminates after 50 ms.
4. Total CPU time of *job1* = 250 ms

The differences in computation speed, architectures, memory speed and other resources affect the expected system performance. Some of the available load-balancer (middle-ware) blindly follow a specific pattern to eradicate these challenges while, others evaluate information obtained from the nodes and base their decision solely on that. Dynamic balancing of load across several heterogeneous

database cluster poses even a major problem when the cluster are located in different database servers geographically or when one database server goes down and the load needs to be distributed to the remaining server.

### ***1.3 Problem Justification***

Prior knowledge of the parameters that are needed to make load balancing more efficient and more manageable is a solution that will enable database developers and administrators to implement the most effective load-balancing strategy. MySQL database has adapted MySQL Proxy as the only application to aid in load balancing, fail over, query analysis, query filtering and modification. However latency, high availability continues to be a major issue while trying to sort out load balancing using MySQL proxy. Also MySQL proxy does not have the capability to give feedback in case there is failure in one of the partition.

Agile software development methodology is the best methodology to be applied when developing the load balancing middle-ware application since it advocates for iterative development and continuous testing at the end of every iterative process. This will facilitate continuous upgrade and improvement of the product during and after development of the load balancing middle-ware and give other researcher who might want to undertake their research in the same area a framework to base their research on.

### ***1.4 Project Objective***

The aim of this research project is to develop a proactive dynamic load balancing middle-ware that provides a software application developer with an implementation that improves performance of transactions in a heterogeneous/federated database management system by performing dynamic load balancing among various connected servers. This will definitely go a long way to contribution of knowledge, value and technologies required to improve database management system performance. The specific objectives of the dynamic load balancing in heterogeneous database partition include:

- i. Design a middle-ware capable of dynamic load balancing.
- ii. Identify and apply a suitable algorithm in developing dynamic load balancing middle-ware.
- iii. Develop a prototype based on the middle-ware.
- iv. Simulate and evaluate the middle-ware.

## ***1.5 Research Outcome***

The outcome of the project undertaking is of major significance because:

We will develop an effective and efficient load balancing middle-ware.

1. The resultant middle-ware will act as a product for further research efforts on database load balancing.
2. It will help in the retention and dissemination efficient load balancer software application.

## ***1.6 Scope***

The study is limited to developing a load balancing middle-ware to be used to balance query load across several heterogeneous MySQL database partitions.

## ***1.7 Limitations***

- i. Simulating and ideal environment.
- ii. Limited material on the actual development of MySQL load balancing applications.
- iii. Availability to various heterogeneous environment to test from.

# Chapter 2: Literature Review

## **2.1 Overview**

Load balancing is to distribute requests to the servers at transport layer, such as TCP, UDP and SCTP transport protocol. The load balancer distributes network connections from clients who know a single IP address for a service, to a set of servers that actually perform the work. Since connection must be established between client and server in connection-oriented transport before sending the request content, the load balancer usually selects a server without looking at the content of the request. After the client request is received by the server the server processes the request and sends the message back to the client acknowledging the client request and the capability to handle the intended job. Load balancing is realized when the most optimal server is chosen to handle the job at hand by a way of voting among an array of servers in a grid.

## **2.2 MySQL Cluster**

While introducing a new service or trying to manage an avalanche of data in real time, your database has to be scalable, fast, and highly available to meet ever-changing market conditions and stringent service-level agreements (SLAs).

MySQL Cluster is considered to be the industry's only true real-time database that combines the flexibility of a high-availability relational database with the low total cost of ownership (TCO) of open source. It features a shared-nothing distributed architecture with no single point of failure to ensure five 9s availability, allowing one to meet their most demanding mission-critical application requirements. Its real-time design delivers consistent millisecond response latency with the ability to service tens of thousands of transactions per second. Support for in-memory and disk-based data, automatic data partitioning with load balancing, and the ability to add nodes to a running cluster with zero downtime enables almost unlimited database scalability to handle most unpredictable workloads [7].

MySQL is an open source ACID (Atomicity Consistency Isolation Durability) compliant Relational Data Base Management System (RDBMS) aiming towards full SQL standards compliance. It has a reputation for ease of use, speed, quality and reliability and consequently is the world's most popular

open source database with over eight million installations.

### 2.2.1 NDB Storage Engine

Storage engines are a unique architectural feature of MySQL. The VFS layer of your operating system allows applications to access files on different file systems through the one interface; MySQLs' storage engine architecture allows applications to access data stored in different ways all through the same SQL interface. Two commonly used storage engines are MyISAM (fast inserts and selects, full text indexes, GIS) and InnoDB (row-level locking, multi-version concurrency, ACID compliant).

MySQL Cluster provides a new storage engine for MySQL. The NDB (also known as ndbcluster) storage engine provides high availability in a shared-nothing architecture. Since there is no shared or special hardware (such as a SAN), MySQL cluster can easily be implemented on affordable commodity hardware. All data is synchronously replicated between nodes. The No Of Replicas configuration parameter dictates how many copies of the data are kept in the cluster.

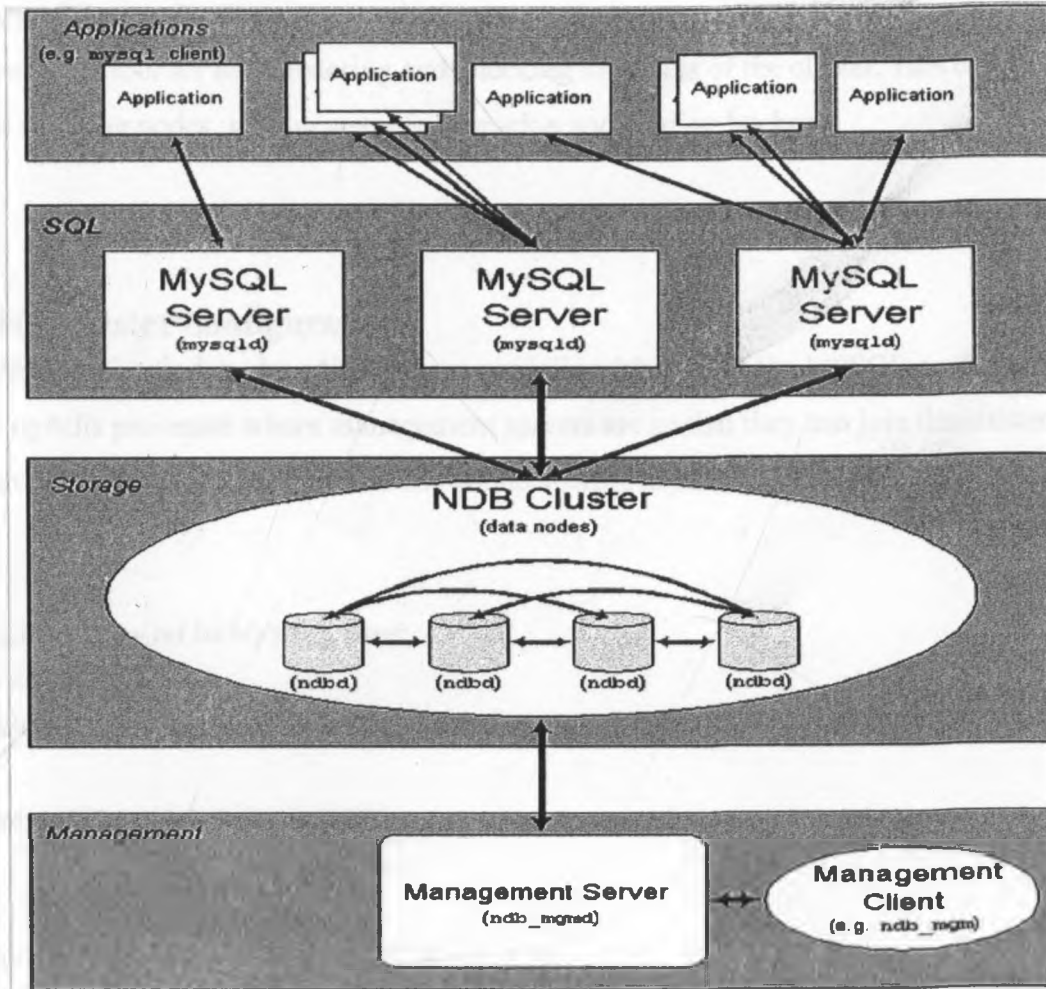
In the 4.1 and 5.0 releases, all data must be held in main memory on the nodes. A rough estimate of the memory needed in each node is:

$$(\text{SizeofDatabase} * \text{NumberOfReplicas} * 1.1) / \text{NumberOfDataNodes}$$

A transaction is committed when it is in memory of more than one node (i.e. it can survive a node crash). The in-memory architecture has the advantage of being very fast, with a single CPU core able to managing over 10,000 transactions per second. Basic persistence is provided by periodically writing checkpoints to disk. The timing between checkpoints (among other things) is configurable. It is also possible to perform on-line backups of data in the cluster. After a system failure (where enough nodes have failed that the cluster no longer has a full data set) the system is restored to the last global checkpoint. When a single node is being restarted (node recovery) it will fetch the latest data from another node in the cluster. The 5.1 release allows non-indexed fields to be stored on disk. In the future, it will be configurable if you want a disk or main memory based cluster. In current releases you can also configure to run in Diskless mode, where no data is ever written to disk (no check-pointing, no logging) [12].

## 2.2.2 MySQL Cluster Components

A basic MySQL cluster implementation looks like this:



*Figure 1: A basic illustration of MySQL cluster setup*

## 2.2.3 MySQL Cluster Implementation

### *Data Nodes (ndbd)*

All data is stored by the data nodes. This data is visible to all the MySQL servers connected to the cluster. Some MySQL special data such as the permissions and stored procedures are not stored in the cluster and must be updated on each MySQL server attached to the cluster.



### ***Management Server Nodes (ndb\_mgmd)***

The management server provides configuration information to nodes joining the cluster. It is not a critical part of the cluster, only needing to be up for a node to join the cluster.

### ***Management Client (ndb\_mgm)***

This is an end-user tool for administering and checking the status of the cluster. This can be used for starting and stopping nodes, getting status information and starting backups.

## **2.2.4 MySQL Cluster Configuration**

The first configuration is done by editing the my.cnf file which holds the MySQL configuration. The connect string tells processes where management servers are so that they can join the cluster. In this case, we have one management server, so the connect string is just a host name.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid from 4.1.8)
# enable ndbcluster storage engine, and provide connectstring
for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com
# provide connectstring for management server host (default
port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com
# provide connectstring for management server host (default
port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com
# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

The Second configuration is done by editing config.ini.config.ini is the cluster configuration file (example below). In this setup we have two data nodes and two replicas. This means that each node holds a complete copy of the database. Here we allow up to three MySQL servers to connect to the cluster.

```
[NDBD DEFAULT]
NoOfReplicas= 2
DataMemory= 500M
IndexMemory= 100M
DataDir= /var/lib/mysql-cluster
[NDB_MGMD]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster
[NDBD]
HostName= ndbd_2.mysql.com
[NDBD]
HostName= ndbd_3.mysql.com
[MYSQLD]
[MYSQLD]
[MYSQLD]
```

### **2.3 MySQL Proxy**

The MySQL Proxy is an application that communicates over the network using the MySQL network protocol and provides communication between one or more MySQL servers and one or more MySQL clients. In the most basic configuration, MySQL Proxy simply interposes itself between the server and clients, passing queries from the clients to the MySQL Server and returning the responses from the MySQL Server to the appropriate client. Because MySQL Proxy uses the MySQL network protocol, it can be used without modification with any MySQL -compatible client that uses the protocol. This includes the MySQL command-line client, any clients that use the MySQL client libraries, and any connector that supports the MySQL network protocol [15].

Two fairly common usage scenarios for MySQL Proxy are:

- i. Load balancing across MySQL slaves.
- ii. Splitting reads and writes so that reads go to the slave database servers and writes go to the master database server.

MySQL Proxy is not necessarily needed to accomplish these goals. For slave load balancing, one can use a regular load balancer in front of your slaves. For read-write splitting, one can use application that uses different DB servers for reads and writes, but that may require significant changes to the application [14].

### 2.3.1 Proxy Scripting Direct Injection

The figure below gives an example of how the proxy might be used when injecting queries into the query queue. Because the proxy sits between the client and MySQL server, what the proxy sends to the server, and the information that the proxy ultimately returns to the client, need not match or correlate. Once the client has connected to the proxy, the sequence shown in the following diagram occurs for each individual query sent by the client [15].

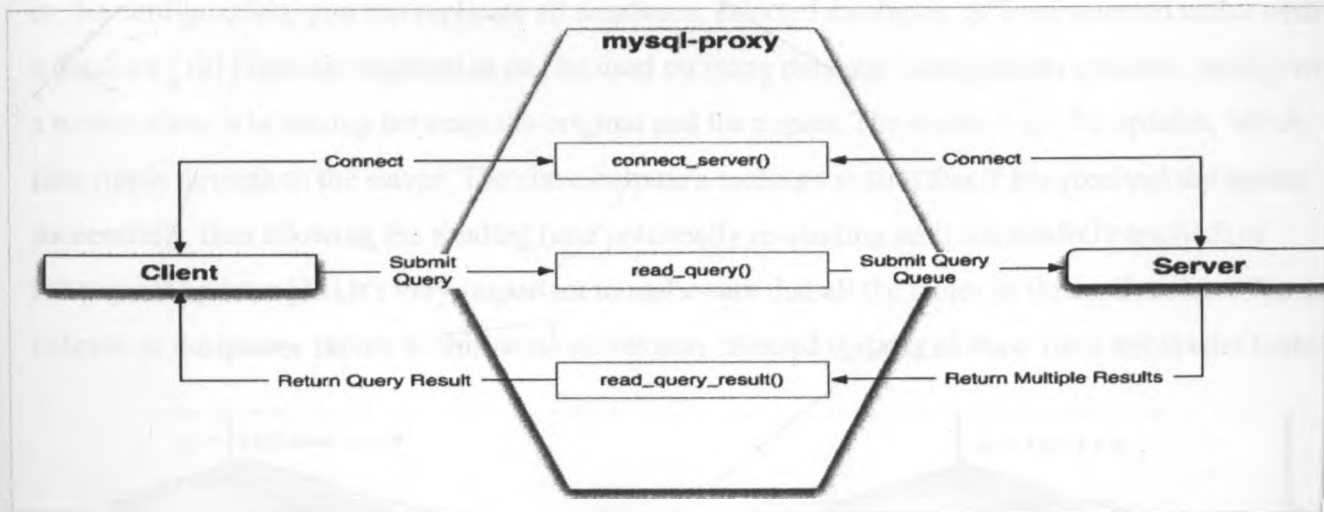


Figure 2: Injection of query in to the query queue.

### 2.3.2 Proxy Load Balancing

Load Balancing selects one backend out of a set of backends to be used as MySQL-server. We use SC (shortest queue first) to distribute the load across the backends equally. Each backend will get the same number of connections.

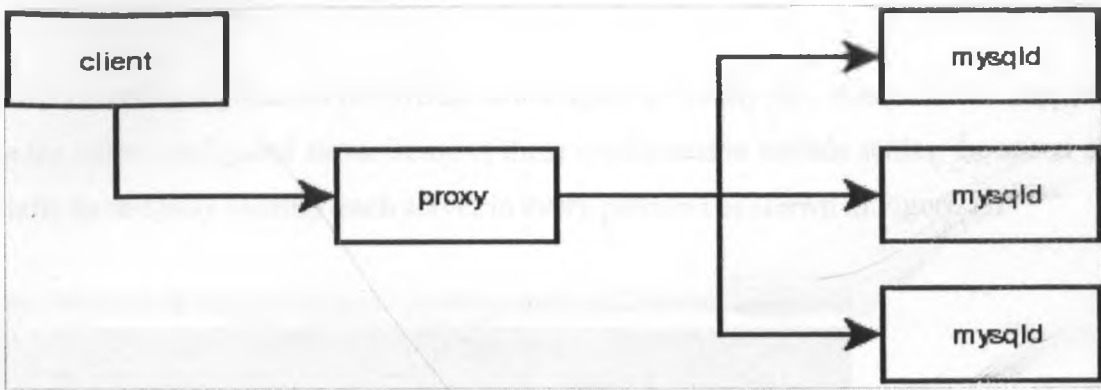


Figure 3: Load balancing using MySQL proxy.

## 2.4 Replication

Replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous - slaves need not be connected permanently to receive updates from the master. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database [12]. Database replication can be used on many database management systems, usually with a master/slave relationship between the original and the copies. The master logs the updates, which then ripple through to the slaves. The slave outputs a message stating that it has received the update successfully, thus allowing the sending (and potentially re-sending until successfully applied) of subsequent updates [13]. It's very important to make sure that all the tables in the replicas have the same indexes as the master failure to this some slaves may rejected updates as done from the master node.

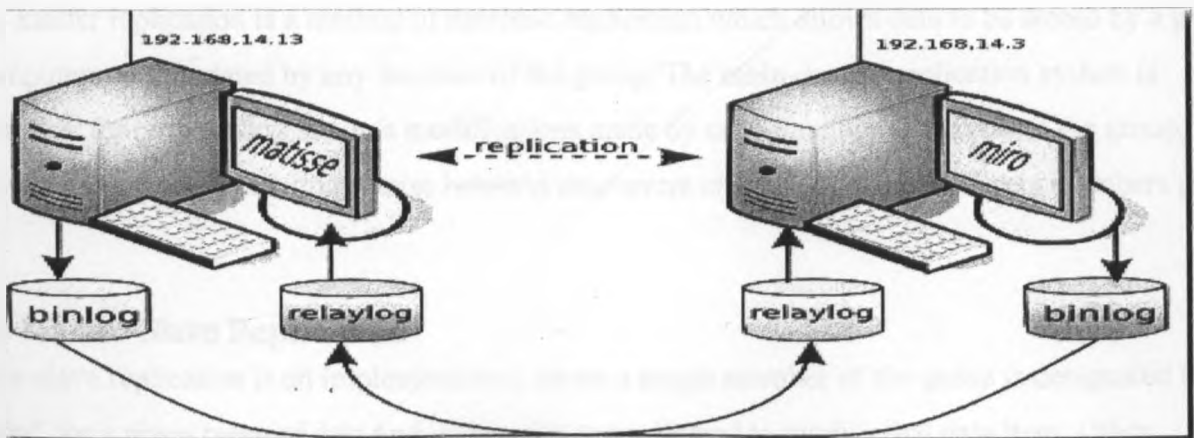
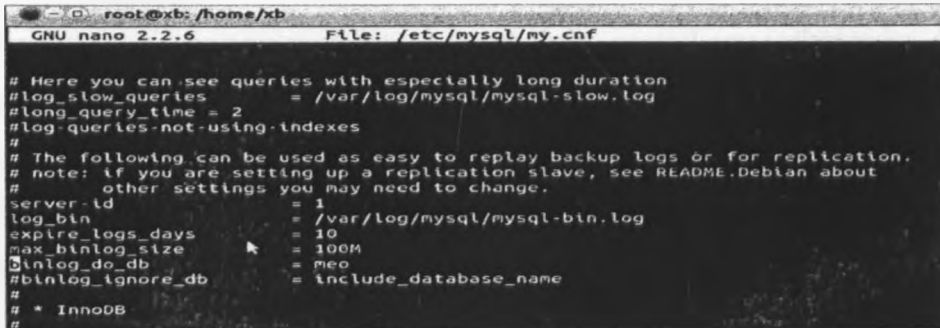


Figure 4: Illustration of a MySQL database replication.

The first step in setting replication in MySQL is configuring the MySQL master which supplies the updates to the entire configured slave. Some of these configuration include setting the server id incrementally to uniquely identify each server in every partition as shown in Figure 1.1

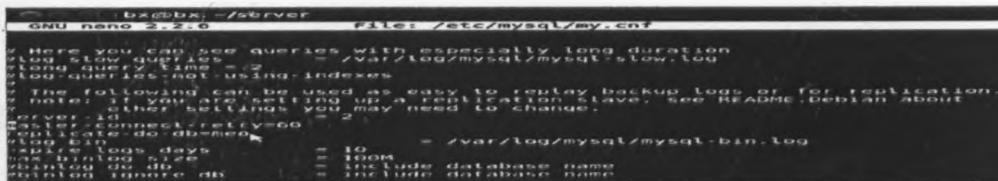


```
root@xb: /home/xb
GNU nano 2.2.6 File: /etc/mysql/my.cnf

# Here you can see queries with especially long duration
#log_slow_queries = /var/log/mysql/mysql-slow.log
#long_query_time = 2
#log_queries-not-using-indexes
#
# The following can be used as easy to replay backup logs or for replication.
# note: if you are setting up a replication slave, see README.Debian about
# other settings you may need to change.
server-id = 1
log-bin = /var/log/mysql/mysql-bin.log
expire_logs_days = 10
max_binlog_size = 100M
binlog-do-db = meo
#binlog-ignore-db = include_database_name
#
# * InnoDB
#
```

Figure 5: Configuration of MySQL Master in my.cnf file.

The second step is to configuring the entire MySQL slave which will be supplied to the updates by the configured master as shown in Figure 1.2



```
b2@bx: ~/server
GNU nano 2.2.6 File: /etc/mysql/my.cnf

# Here you can see queries with especially long duration
#log_slow_queries = /var/log/mysql/mysql-slow.log
#long_query_time = 2
#log_queries-not-using-indexes
#
# The following can be used as easy to replay backup logs or for replication.
# note: if you are setting up a replication slave, see README.Debian about
# other settings you may need to change.
server-id = 2
master-host = meo
master-user = meo
master-conn-opts =
log-bin = /var/log/mysql/mysql-bin.log
expire_logs_days = 10
max_binlog_size = 100M
binlog-do-db = include_database_name
binlog-ignore-db = include_database_name
#
# * InnoDB
#
```

Figure 6: Configuration of MySQL Slave in my.cnf file.

### 2.4.1 Multi-Master Replication

Multi-master replication is a method of database replication which allows data to be stored by a group of computers, and updated by any member of the group. The multi-master replication system is responsible for propagating the data modifications made by each member to the rest of the group, and resolving any conflicts that might arise between concurrent changes made by different members [17].

### 2.4.2 Master-Slave Replication

Master-slave replication is an implementation where a single member of the group is designated as the "master" for a given piece of data and is the only node allowed to modify that data item. Other members wishing to modify the data item must first contact the master node. Allowing only a single

master makes it easier to achieve consistency among the members of the group, but is less flexible than multi-master replication [17].

## **2.5 Load Balancing**

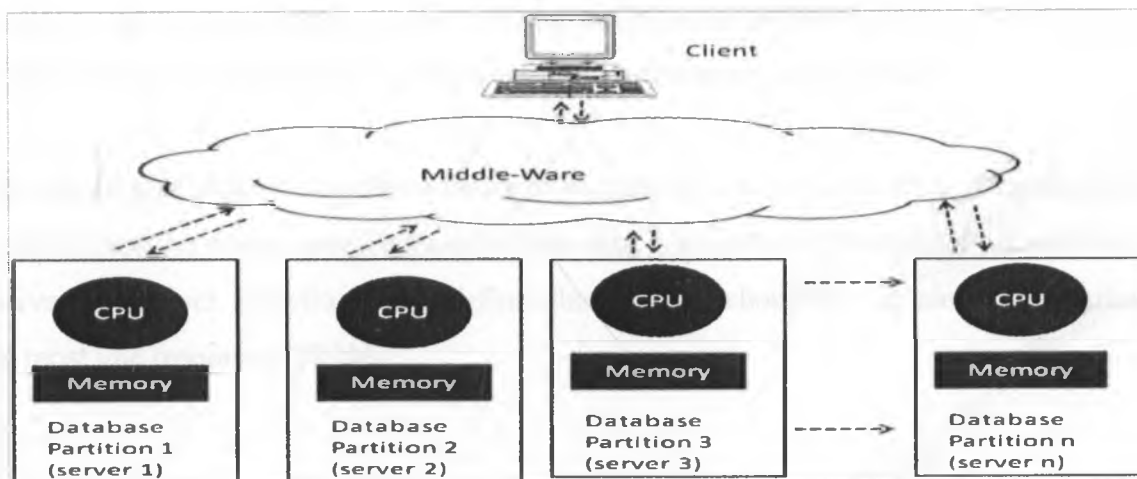
Load balancing is a computer networking methodology to distribute workload across multiple computers or a computer cluster, network links, central processing units, disk drives, or other resources, to achieve optimal resource utilization, maximize throughput, minimize response time, and avoid overload. Using multiple components with load balancing, instead of a single component, may increase reliability through redundancy. The load balancing service is usually provided by dedicated software or hardware, such as a multilayer switch or a Domain Name System server [12].

### **2.5.1 Static Load Balancing**

In static load balancing work is initially partitioned among the processors using some heuristic cost function, and there is no subsequent data or computation movement to correct load imbalances which result from the dynamic nature of mining algorithms [8].

### **2.5.2 Dynamically Load Balancing**

Dynamic load balancing seeks to address this by stealing work from heavily loaded processors and re-assigning it to lightly loaded ones. Computation movement also entails data movement, since the processor responsible for a computational task needs the data associated with that task as well. Dynamic load balancing thus incurs additional costs for work/data movement, but it is beneficial if the load imbalance is large and if load changes with time. Dynamic load balancing is especially important in multi-user environments with transient loads and in heterogeneous platforms, which have different processor and network speeds. These kinds of environments include parallel servers, and heterogeneous, meta-clusters. Dynamic load balancing algorithms make changes to the distribution of work among workstations at run-time; they use current or recent load information when making distribution decisions [8].



**Figure 7 : Dynamically load balancing in a heterogeneous database partition.**

### 2.5.3 Load Balancing Strategies

There are three major parameters which usually define the strategy a specific load balancing algorithm will employ, which are important in order to address issues such as who makes the load balancing decision, what information is used to make the load balancing decision and where the load balancing decision is made. When concentrating on selecting a policy such that in policy selection, information gathering policy specifies the strategy for the collection of load information includes the frequency and method of information gathering. Information policy specifies what workload information to be collected, from where it is to be collected. The frequency is determined based on a tradeoff between the accuracy of load information and the overhead of information collection.

**Initiation Policy-** determines who starts the load balancing process. The process can be initiated by an overloaded server (sender-initiated) or by an under-loaded server (receiver-initiated). Sender initiated policies are those where heavily loaded nodes search for lightly loaded nodes while receiver initiated policies are those where lightly loaded nodes search for suitable senders [9][11].

**Job Transfer Policy-** determines when job reallocation should be performed and which job(s) should be reallocated. Job reallocation is activated by a threshold based strategy. In a sender-initiated method, the job transfer is invoked when the workload on a node exceeds a threshold. In a receiver-initiated method, a node starts the process to fetch jobs from other nodes when its workload is below a threshold [9][11]. The threshold can be a pre-defined static value or a dynamic value that is assessed at runtime

based on the load distribution among the nodes. When job reallocation is required, the appropriate job(s) will be selected from the job queue and transferred to another node.

**Resource type policy-** classifies a resource as a server or receiver of a task according to its availability status. Location policy uses the results of resource type policy to find who work co-ordingly with server or receiver. Selection policy defines the tasks that should be migrated from overloaded resources to most idle resources [9][11].



# Chapter 3: Methodology and Design

System design methods are a discipline within the software development industry which seeks to provide a framework for activity and the capture, storage, transformation and dissemination of information so as to enable economic development of computer systems that are fit for purpose.

In our study we used the Agile software development methodology based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle [4]

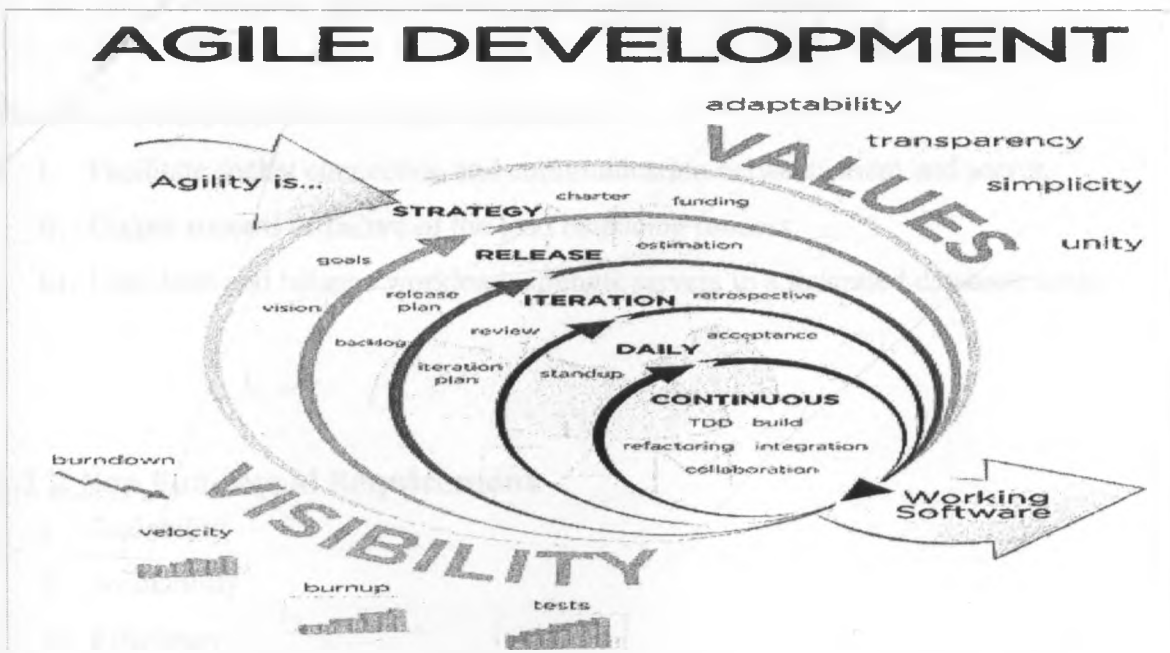


Figure 8: Agile software development methodology.

## **3.1 Project Requirement**

### **3.1.1 Functional Requirement**

The client part of the middle-ware should be able to:

- i. Read from a file and accept an array of IP addresses representing the database nodes.
- ii. Send a request to every server node in the list.
- iii. Receive a success or failure and server usage parameters message from the server.
- iv. Compute the returned values and choose the most optimal server based on parameters returned.
- v. Send the database query to the most optimal server.

The Server part of the middle-ware should be able to:

- i. Receiver a request from the client.
- ii. Compute:
  - (a) Memory usage.
  - (b) CPU Usage.
  - (c) Check if mysqld is running.
- iii. Send a success or failure message and parameters computed.

The Middle-ware as a whole should be able to:

- i. Facilitate socket connection and communication between client and server.
- ii. Output success or failure of the load balancing process.
- iii. Distribute and balance workload amongst servers in a federated database setup.

### **3.1.2 Non Functional Requirement**

- i. Scalability
- ii. Availability
- iii. Efficiency

### **3.1.3 Non Functional Requirement**

#### **Hardware:**

- i. Laptop.
- ii. Network switch.

#### **Software:**

- i. Ubuntu operating system.
- ii. Eclipse IDE.
- iii. MySQL J connector.
- iv. MySQL database.

## **3.2 System Architecture**

### **3.2.1 Middle-Ware Architecture**

The process starts with a user request from the client side, the client reads a file storing the possible IP addresses representing each representing a servers/nodes and loops through each address. As the client loops through the IP address it creates a socket connection with an intent to connect to the server. The server accepts or rejects a socket connection from the client and initiates the process of computing percentages of:

- i. CPU usage.
- ii. Memory Usage.
- iii. Check if mysqld is running.

Once the server is done computing the resources usage, the server sends back an aggregated message of all computed values and return them in an arraylist. The client on the other side receives feedback message from each server and get the time difference between the time of socket initiation to the time the server returned a success message to determine the response time/network throughput. The client performs a comparison to establish which server has the least/optimal usage of resources. The most optimal amongst the server is chosen by the client and that's where the client sends the database query.

### Dynamic Load Balancing Middle-ware

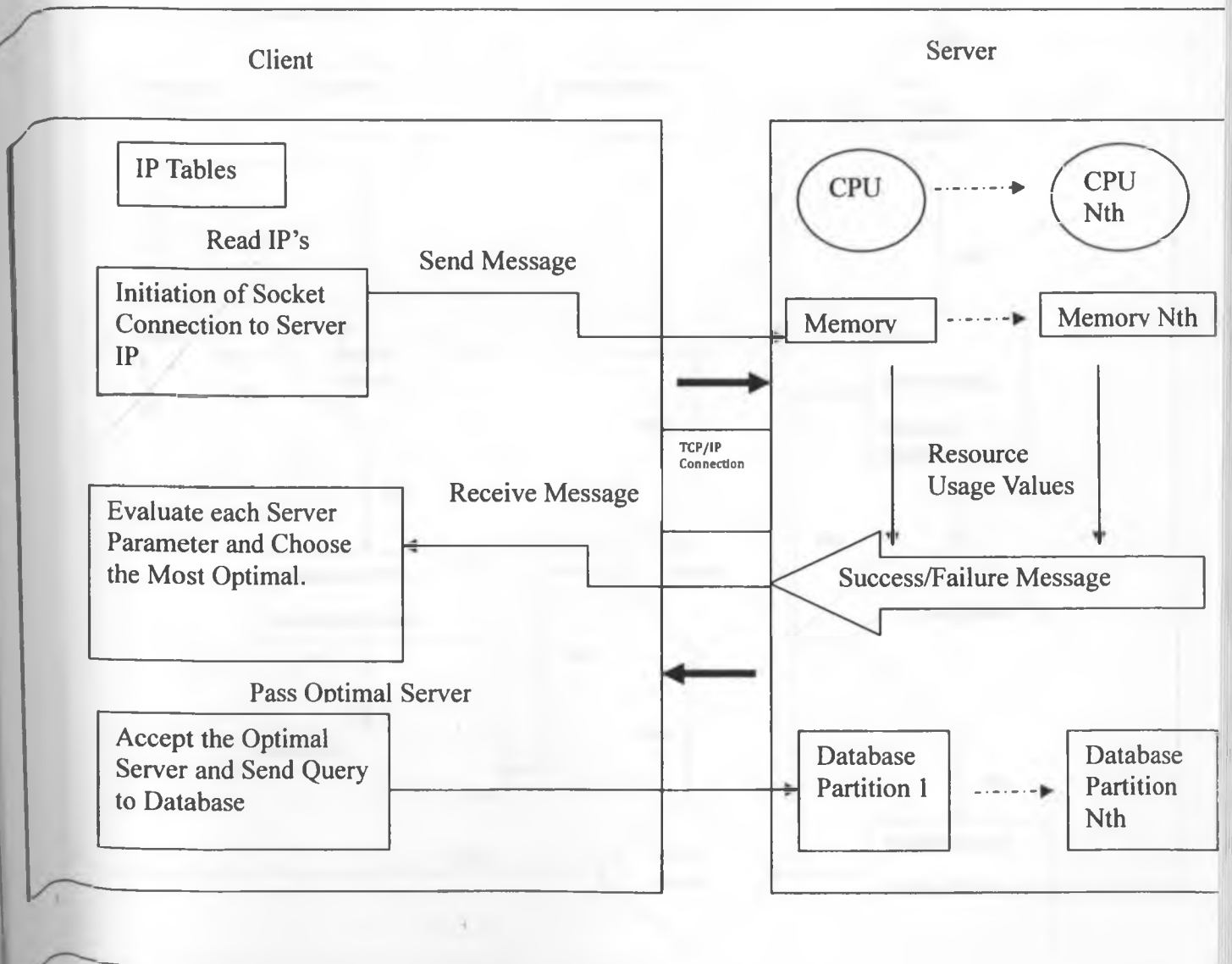


Figure 9 : Load Balancing Middle-ware Architecture.

### 3.2.2 Process Flow

The process flow represent the interaction of components and the way information flows from one component to the other.

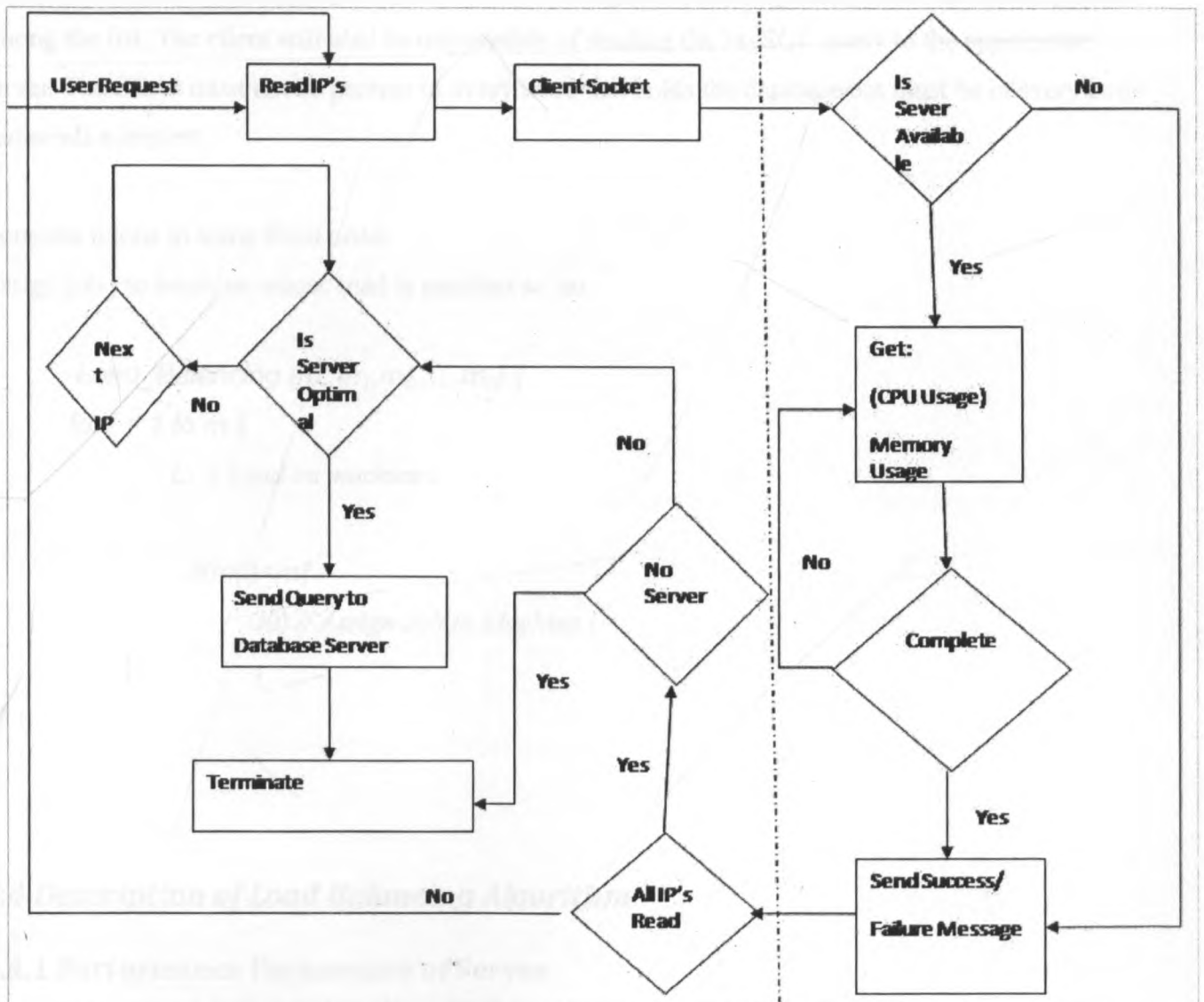


Figure 10: Load balancing middle-ware process flow.

### **3.3 Theoretical Analysis of Load Balancing**

We will achieve our objectives by using Java socket oriented middle-ware with two Java classes, the ClientSocket Class and the ServerSocket Class. The ServerSocket Class will run in every node that holds the database partition and it's responsible for collecting the node resource usage and sending them back to the client. On the other hand the ClientSocket Class will be responsible for initiating the request to the server and aggregating all the server computed parameters and chose the most optimal among the list. The client will also be responsible of sending the MySQL query to the appropriate server. The Client must not be present in every node that holds the database but must be in every node that sends a request.

Consider n jobs in some fixed order.

Assign job j to machine whose load is smallest so far.

```
Load_Balancing (m, m1, m2, ..... mn) {
```

```
for i = 1 to m {
```

```
    Li // Load on machine i
```

```
    if(m(i) < m {
```

```
        J(i) // Assign Job to Machine i
```

```
    }
```

```
}
```

### **3.4 Description of Load Balancing Algorithm**

#### **3.4.1 Performance Parameters of Server**

In order to achieve higher system throughput and shorten the client's feedback time, the algorithm uses dynamic parameters to reflect the capability of the server. During the processing of the client-server

system, each server's load is changing as time goes on, the system has to estimate the load-balance according to the real-time server load, and these are called dynamic performance parameters. This report selected the following dynamic performance parameters:

- (1) Processor utilization ratio: it can reflect the busyness degree. The process server monitor inspects the CPU utilization ratio, so as to confirm the CPU's load.
- (2)Memory utilization ratio: the size of the server memory changes as the system runs. The process of server monitor inspects the utilization ration of physical memory, so as to confirm the server memory's load.
- (3)Network Throughput: the network data are mainly transferred through TCP mode in client-server system, the process of sever monitor inspects the time taken from the time of client request to the time the server responded.

### 3.4.2 Performance Parameters Optimization

To estimate the performance parameters .i.e. CPU load, memory usage and network throughput we use `getSystemLoadAverage` java function which is derived from `OperatingSystemMXBean` interface. The `getSystemLoadAverage` returns the system load average for the last minute. The system load average is the sum of the number of runnable entities queued to the available processors and the number of runnable entities running on the available processors averaged over a period of time. The way in which the load average is calculated is operating system specific but is typically a damped time-dependent average. If the load average is not available, a negative value is returned.

Memory usage is estimates by getting the total memory, used memory and free memory using the java `Runtime` class. This is achieved by subtracting total memory from free memory to get committed memory.

Total = total Memory

Free =free Memory

Used = total – free

After that one can get the percentage of memory usage as:

$(\text{used}/\text{free}) * 100.$

Network throughput is estimated by getting the difference in millisecond from the time the request was initiated by the client to the time the server returns a success message back to the client.i.e

Network Throughput= CurrentSystemTime-Elapsetime.

### 3.4.3 The Load Balancing Algorithm

#### Client Algorithm

```
Client_Connect ( ) {  
  For each server i in List  
    Initial_SystemTime=Get_SystemTime()  
    //attempt connection to the server  
    Server_Messenger[i]  
      if request R(i) responses then{  
        //get time fpr the server tp respond  
        Time_taken=Get_SystemTime()-Initial_SystemTime  
        //Set load for server i  
        Server[i] = load[i]+Time_taken  
      else  
        //Server not reachable  
      }  
  index=0  
  // j is server with load  
  for each j in server {  
    //Check if server returned  
    if j is 0 {
```



```

index=server[j] //Load in server
}
//Find the most optimal server
if index is less than server[j] {
index= server[j] //server[j] is the most optimal
else
index is the optimal server
}
//Connect to database with the optimal server
database_Connect(index)
Initial_SystemTime=Get_SystemTime()
Fetch_Data() from index
//Get fetch time taken
ElapsedTime=Get_SystemTime()-Initial_SystemTime

} //end client

```

### Server Algorithm

```

Server_Connect{
Receive_ClientMessage()
//Get server resource usage
if message is success{
get_Cpu_Usage()
get_Memory_Usage()
}
//CPU resource usage
get_Cpu_Usage() {
ThreadTime=Create Threads to read multi core processors processing time
get Processor load()

```

```

factor=get Processor load()+ThreadTime
return factor
}
//Memory resource usage
get_Memory_Usage() {
total_Memory=total server memory
Used_Memory=current memory in use
factor=total_Memory-total_Memory
return factor
}
FeedBack=get_Cpu_Usage(), get_Memory_Usage()
Send_ClientFeedback(FeedBack)
}

```

### **3.5 Implementation**

The implementation phase seeks to put into action the discussions in the requirements and the design phases.

#### **3.5.1 Development environment**

- i. Ubuntu 11.10 (Oneiric Ocelot).
- ii. Eclipse Indigo.
- iii. JAVA Programming Language
- iv. MySQL database.
- v. Mysql-connector-java-5.1.18-bin.jar.

#### **3.5.2 Implementation tools**

Java language was used as the programming language of choice and MySQL database as the database of choice.

### 3.5.3 Why JAVA and MySQL Databases

The load balancing middle-ware was developed using Java programming language. The rationale for selecting Java as the development language is that;

- a) Java Sockets-A socket is a software endpoint that establishes bidirectional communication between a server program and one or more client programs. The socket associates the server program with a specific hardware port on the machine where it runs so any client program anywhere in the network with a socket associated with that same port can communicate with the server program [5]. Thus the Java socket will facilitate connection and communications between the partitions, middle-ware and the requesting application.
  
- b) JDBC (Java Database Connectivity)-a Java API that enables Java programs to execute SQL statements. This allows Java programs to interact with any SQL-compliant database. Since nearly all relational database management systems (DBMSs) support SQL, and because Java itself runs on most platforms, JDBC makes it possible to write a single database application that can run on different platforms and interact with different DBMS s [6].

MySQL database was selected since load balancing has not been extensively studied for MySQL databases compared to other proprietary databases like oracle and MS-SQL server thus making MySQL the most viable choice.

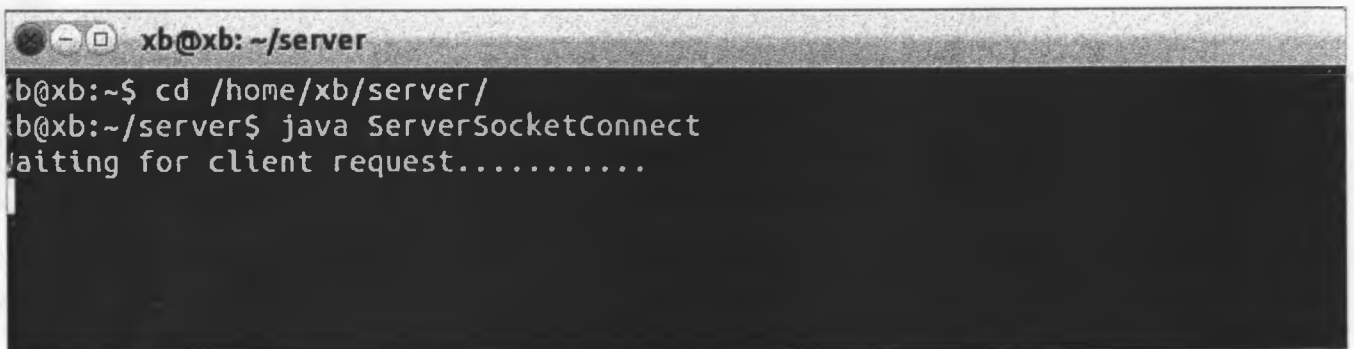
# Chapter 4: Results Analysis

## 4.1 Discussion of Results

### 4.1.1 System Result

The dynamic load balancing system for heterogeneous database system is able to balance loads among several federated databases partitions located on different servers. It computes the most optimal server among an array of servers and sends the database query to the most optimal amongst them. This show that the load-balancer can be able to scale up and down, ensure availability of data and increase the response time. The diagrams below show the outputs of the client and the server.

#### Server Listener



```
xb@xb: ~/server
xb@xb:~$ cd /home/xb/server/
xb@xb:~/server$ java ServerSocketConnect
Waiting for client request.....
```

Figure 11: Server listener waiting for a client connection.

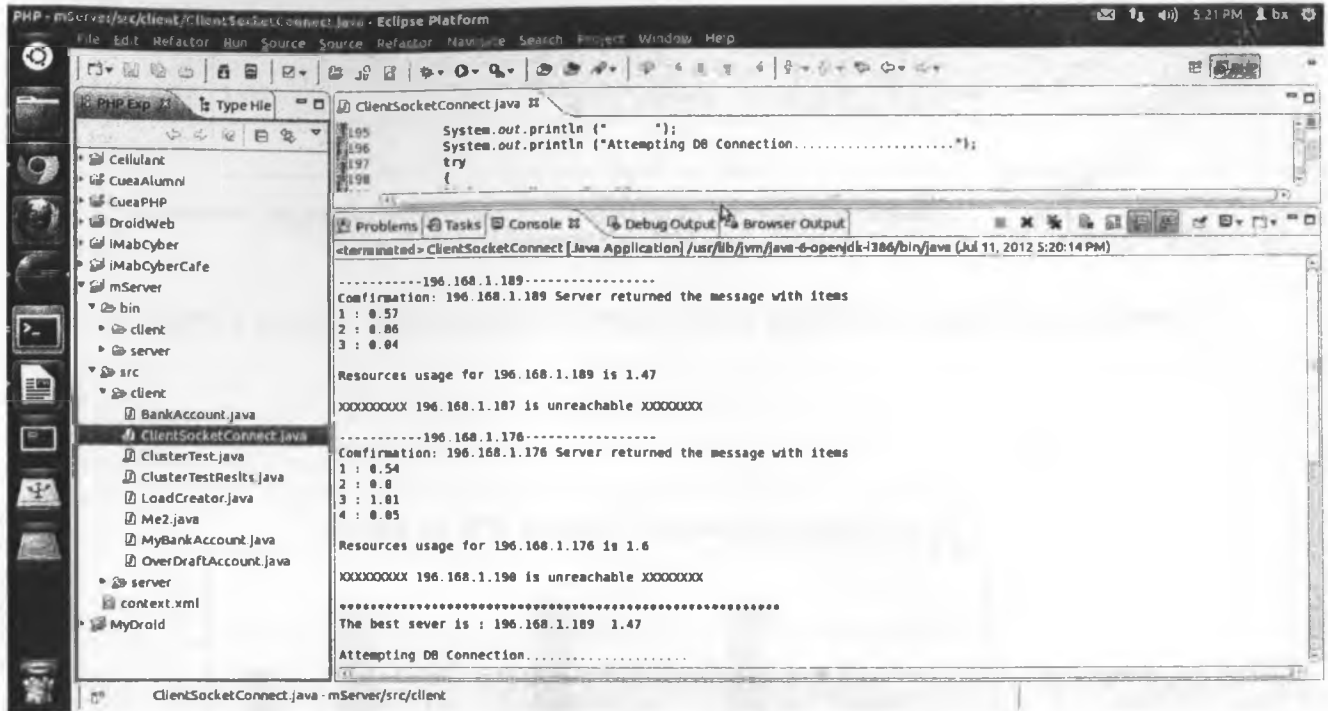
#### Server Computation.



```
xb@xb:~/server
xb@xb:~$ cd /home/xb/server/
xb@xb:~/server$ java ServerSocketConnect
Waiting for client request.....
Clients Message: Confirmation : Client Sent a request
*****MEMORY RESOURCES*****
Total Memory: 61210624
Used: 118912
Free: 61091712
Percent Used: 0.19426692987152033
Percent Free: 99.80573307012848
-----
*****CPU RESOURCES*****
-5237961366719996138 ... Thread[Thread-2,5,main]: cpuTime = 160000000
-5237961366719996138 ... Thread[Thread-1,5,main]: cpuTime = 400000000
-5237961366719996138 ... Thread[Thread-3,5,main]: cpuTime = 160000000
-5237961366719996138 ... Thread[Thread-5,5,main]: cpuTime = 400000000
-5237961366719996138 ... Thread[Thread-4,5,main]: cpuTime = 100000000
Total elapsed time 133098/203
Total thread CPU time 1880000000
Factor: %.2f%n 1.4124854061425563E-9
My processors 2
My Load 0.03
Waiting for client message...
```

Figure 12: Server computations on memory usage and CPU usage.

## Client Computations



The screenshot shows the Eclipse IDE with the following components:

- Project Explorer:** Shows a project named 'mServer' with a sub-package 'client' containing 'ClientSocketConnect.java'.
- Editor:** Displays the source code for 'ClientSocketConnect.java' with line numbers 195-199. The code includes a try block for database connection.
- Console:** Shows the execution output for 'ClientSocketConnect [Java Application]'. It displays confirmation messages for servers at 196.168.1.189 and 196.168.1.176, resource usage statistics, and unreachable server warnings for 196.168.1.187 and 196.168.1.190. It concludes with 'The best sever is : 196.168.1.189 1.47' and 'Attempting DB Connection...'.

Figure 13: Client computations and results.

### 4.1.2 Test Results

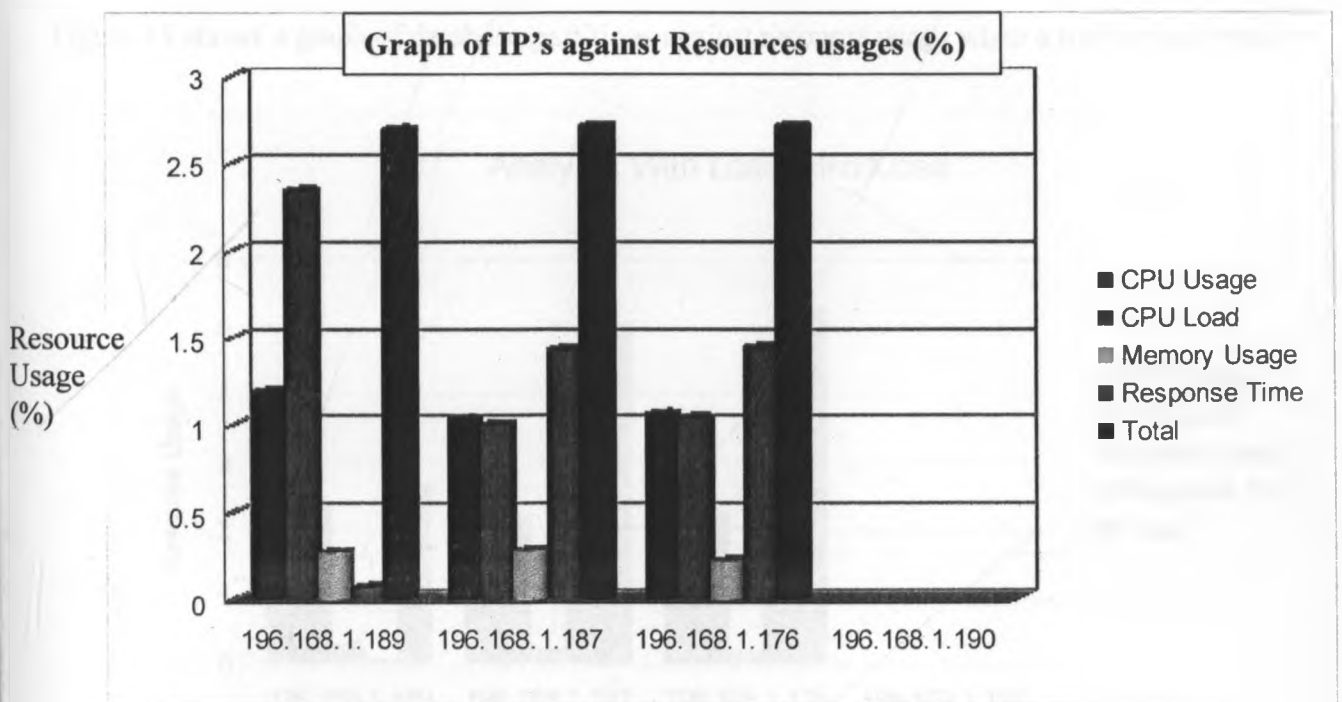
The program was tested using trial test case of the actual happening where several laptop were setup connected via a TCP-IP environment. There was one laptop (but not limited to one) acting as the client and other acting as the server. The cable was unplugged on one machine to test if fail-over was implemented. The program was also tested without load and load introduced, for testing purposes the load that was introduced was a non ending continuous for loop that affects computer resources usage like memory, CPU usage and hence affecting response time.

**Testing while no load introduced to any server.**

	CPU Usage (%)	CPU Load (%)	Memory Usage (%)	Response Time (%)	Total (%)
196.168.1.189	1.19	2.34	0.27	0.07	2.69
196.168.1.187	1.03	1.0	0.28	1.43	2.71
196.168.1.176	1.06	1.04	0.22	1.44	2.71
196.168.1.190		0	0	0	

**Table 1: Resource usage for all the servers when load is not introduced.**

Figure 14 shows a graph of servers against resource usage when there is not load introduced.



**Figure 14: Graph showing results for each IP/Server without load introduced.**

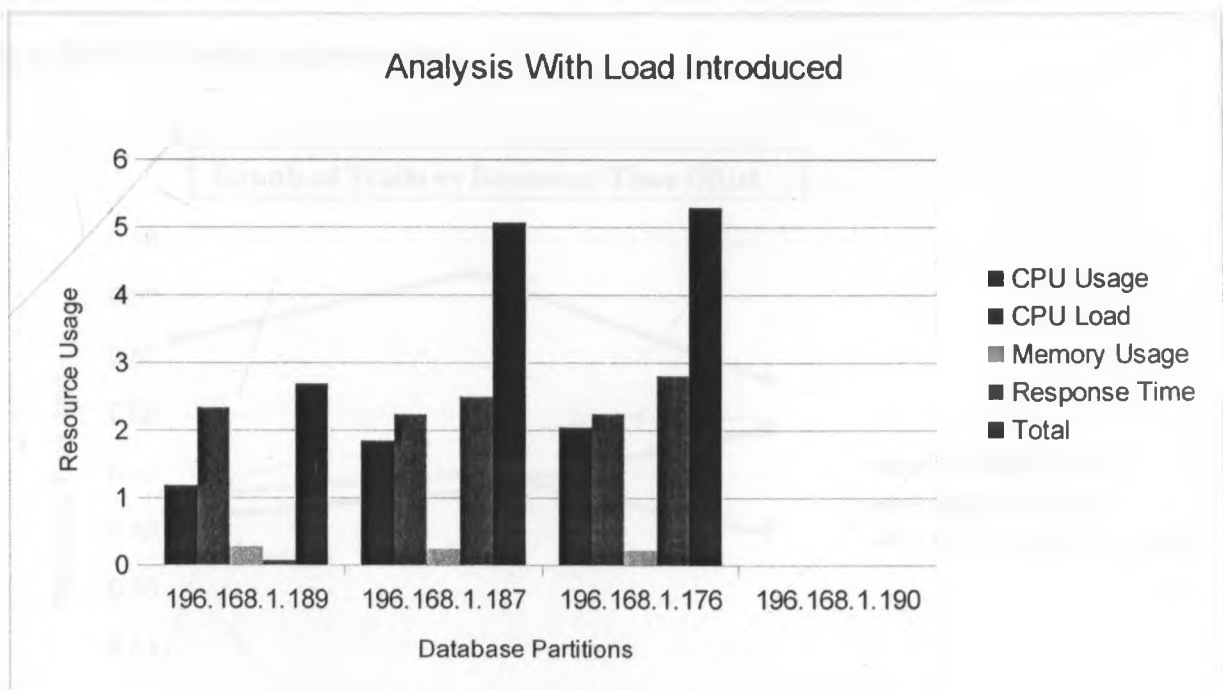
In figure 14 the analysis of total resource usage when no load is introduced in any of the server shows that the total load for all the server falls well below 3 this is and indicator that the middle-ware is able to monitor all the resources in each server and communicate the right values of each resource. The similarity in deviation of total load for each server creates a clear image that every server is monitored without bias.

**Test while load is introduced on all servers.**

	CPU Usage (%)	CPU Load (%)	Memory Usage (%)	Response Time (%)	Total (%)
196.168.1.189	1.19	2.34	0.27	0.07	2.69
196.168.1.187	1.85	2.24	0.24	2.5	5.07
196.168.1.176	2.05	2.25	0.22	2.81	5.29
196.168.1.190	0	0	0	0	0

**Table 2: Resource usage for all the servers when load is introduced.**

Figure 15 shows a graph of database partitions against resource usage when a load is introduced.



**Figure 15: Graph showing results for each IP/Server with load introduced.**

Figure 15 shows the analysis of the same database partitions servers but this time loads are introduced on server 196.168.1.187 and 196.168.1.176 only leaving out server 196.168.1.189. The experiment again shows very clearly that the total resource usage for the two servers (196.168.1.187 and 196.168.1.176) goes beyond 3 while server 196.168.1.189 remains almost the same. This is a clear

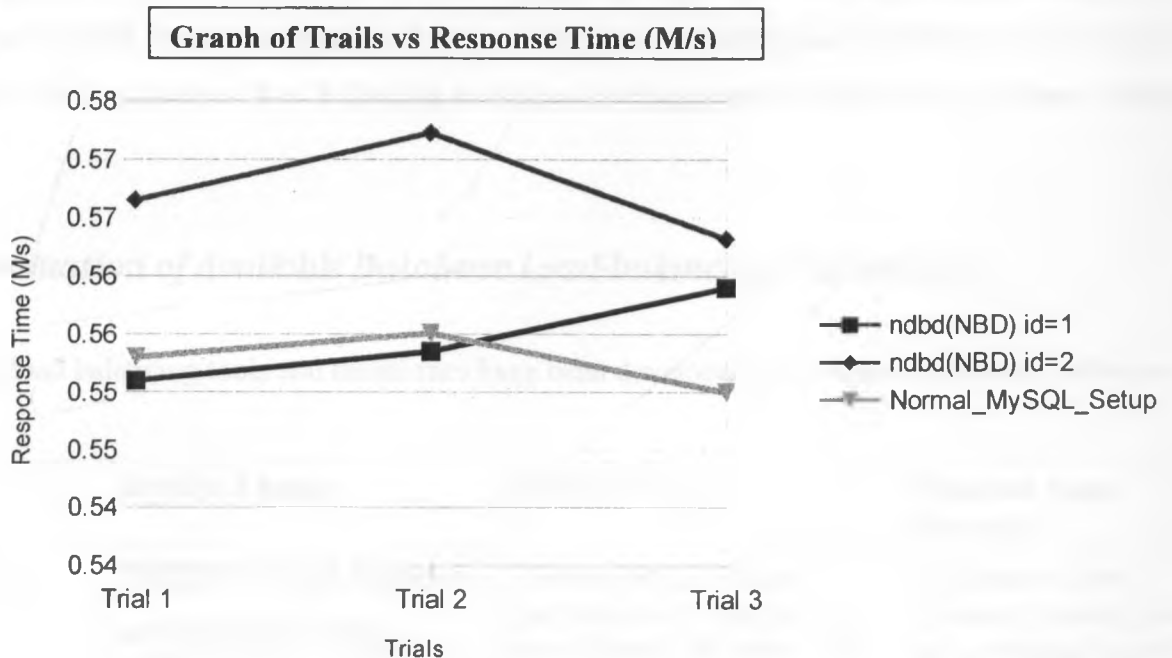
indication that the middle-ware is capable of detecting any slightest change in resource usage and communicates the results without bias.

### MySQL cluster vs Dynamic Load balancer.

The tables below shows a comparison test carried out to test the query response time, between query from a MySQL cluster setup and querying direct from a normal MySQL setup as it would be done by the Dynamic Load-balancing middle-ware. This was achieved by querying the database severally and capturing the response time for each attempt.

	ndbd(NBD) id=1 Response Time(M/s)	ndbd(NBD) id=2 Response Time(M/s)	Normal_MySQL_Setup Response Time(M/s)
Trial 1	0.5510	0.5665	0.5530
Trial 2	0.5535	0.5723	0.5551
Trial 3	0.5590	0.5632	0.550

**Table 3: Trials for nodes response time.**



**Figure 16: A graph of trials against response time in millisecond.**

Figure 16 show a graph of response time against several trials and from the results we can see that d



node ndbd(NBD) id=2 takes more time to respond since it's a slave to node ndbd(NBD) id=1 while node ndbd(NBD) id=1 and Normal\_MySQL\_Setup takes a relatively faster time than node ndbd(NBD) id=2. This is because the MySQL cluster management node (ndn\_mgmd) has to vote which node will respond to the request which means more response before the best node is chosen.

#### **4.2 Challenges Facing Database Querying in Federated Database.**

Database querying involves fetching hundreds of millions of rows of data stored in a database management system. The amount of time taken by the DBMS to query the data is even complicated by the hardware limitations to handle the execution load. In federated database environment the problem is further complicated considering that now you have to deal with not only the computation capability of the server but also the network interconnecting different federated databases.

Most database management systems have tried to come up with applications that try to distribute the load among several database partitions to reduce the amount of time taken between querying transactions. However these solutions are still limited in their quest to balance between scalability, availability, and response time and network throughput. Some of the existing systems have great response time but low availability while others have great scalability and availability but low response time thus making database load balancing as the preferred approach to solve these problems effectively.

#### **4.3 Evaluation of Available Database Load-balancing Techniques.**

Several load balancing tools and techniques have been developed over times as summarized below.

	<b>MySQL Cluster</b>	<b>MySQL Proxy</b>	<b>Dynamic Load-balancer</b>
<b>Fail-Over</b>	Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits	There is not intelligent mechanism of handling fail-over in MySQL proxy such as telling if one server is down go to the next one[22] what is called fail-over in MySQL proxy is the capability of giving a notification message that the	In dynamic load-balancing middle-ware the emphasis is put the server as a unit of computation hence the only thing the middle-ware needs to concern its self with is the resource usage of that

	for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable [19].	server chosen is down.	particular server and return it as either the most optimal server or not. This was fail-over reduced drastically.
<b>Machine architecture</b>	All machines used in the cluster must have the same architecture. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine [19].	Just like the dynamic load-balancer middle-ware MySQL proxy is not dependent on any machine architecture as long as MySQL database is running in that machine [15].	The machine architecture is not of importance in dynamic load-balancer as long as MySQL is running in any particular server the load balancer will choose any of the server as long as it's the most optimal
<b>Scalability</b>	MySQL cluster is very efficient in scalability but it's limited to 1-63 nodes maximum [21].	MySQL proxy achieves scalability by using LUA scripting with is a query injecting mechanism where a single request is issued by data is queried from different servers and merged together as if the data was queried from a single database server[14].	Dynamic load-balancer middle-ware doesn't restrict the number of server that can be added to the queue but the limitations can come due to client's computation capabilities.
<b>Response Time</b>	Response time is quite commonly worse in MySQL cluster than with the traditional since management has to be done by the cluster manager (NDB) which uses sequential access to NDB storage engine [20].	There is no emphasis of response time in MySQL proxy since querying is done on all the existing server and hence response time is pegged on the network throughput if it's high the response time will be high and vice versa[15].	In dynamic load-balancer middle-ware emphasis is put on both scalability and response time. This is achieved by choosing the most optimal server with response time as one of the parameter.
<b>Database architecture</b>	MySQL cluster is limited to	Just like MySQL cluster the MySQL proxy is limited to	Although the case study of this report was

	<p>MySQL databases only.</p> <p>Other database has their own way of load balancing thus making MySQL clustering as a narrow solution.</p>	<p>operate on MySQL databases only.</p>	<p>limited to MySQL databases the dynamic load-balancer middle-ware can be easily applied to any other database architecture.</p>
--	---	---	---

**Table 4: Evaluation of various load balancing techniques.**

### 4.3.1 Analysis of different load balancing techniques.

Based on the above sampled evaluations one can make a conclusion that there is no specific approach that can be used to solve all aspects of load balancing single handily but some implementation are better suited to offer a wide range of solution that others. In this case the Dynamic Load-balancer Middle-ware fairs well in the way it handles fail-over, response time, compatibility to different machine architectures and appreciation of different database architectures. Although scalability is better handled by MySQL cluster its limitation of maximum of 63 nodes [21] makes the Dynamic Load-balancer Middle-ware an idea choice when it comes a federated databases environment that may require more partitions to use. In a MySQL cluster setup response time can be delayed if the management node(ndb\_mgmd) is also acting as the SQL node as well as a data node further more total unavailability can occur if management node goes down. The figure below shows a MySQL cluster setup.

```

Administrator: C:\Windows\system32\cmd.exe - ndb_mgm
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\bx>cd \
C:\>cd mysql\cluster
C:\mysql\cluster>ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration

[ndbd(NDB)] 2 node(s)
id=2  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4, Nodegroup: 0, Master>
id=3  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4, Nodegroup: 0>

[ndb_mgmd(MGM)] 1 node(s)
id=1  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4>

[mysqld(API)] 3 node(s)
id=4  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4>
id=5  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4>
id=6  @127.0.0.1  <mysql-5.5.19 ndb-7.2.4>

ndb_mgm>

```

**Figure 17: MySQL cluster test environment.**

# Chapter 5: Conclusion

## **5.1 Achievement**

In this study we were able to:

- i. Develop a middle-ware that balances database query load among several federated database partitions located in a heterogeneous environment.
- ii. Discuss several strategies of database load balancing their weakness and strength.
- iii. Develop a client-server implementation that facilitates interactions between distributed database servers and clients.
- iv. Testing and evaluation of the middle-ware effectiveness.

## **5.2 Challenges**

This study was faced with several challenges which among them include:

- i. The testing environment was a mimic of a real environment, testing with a real environment meant asking for companies that have large distributed database environment setup which was not possible due to data security.
- ii. Database load balancing is not extensively studied especially in MySQL databases which makes documentation and literatures scarce.

### **5.3 Further Study**

The result of this study was very promising and it's an indicator that more research need to be carried on this area due to the inherent growth of more complex database architectures. This shows there is need to keep a breast with those complexities otherwise they may delay service delivery by the information technology industry.

Some of the areas that were identified for further development include:

- i. Develop a load-balancer middle-ware that takes in account the disk read-write speed.
  
- ii. Develop a middle-ware that can balance load between different database e.g Oracle, MySQL,MS-SQL e.t.c.s

### **5.4 Conclusion**

This report points out a load balancing strategy to be adopted to balance load in federated database environment. It thus focuses on the emphasis of improved performance, speed and data availability even in conditions that would seem challenging to achieve the same. The dynamic load balancer developed during this research project will be of great benefit especially to developers who wish to develop an application that connects to a database management system. This is because they will be able to implement the load balancer in their application development as an add-on to tweak their system performance. This study will also add more knowledge to future developer's who wish to engage in the area of database development

Also having the best load balancing techniques implemented from the point of view of a user translates to improved effectiveness and efficiency on ways of conducting business. For big companies it can be seen as a way of gaining a competitive edge over a competitor who is yet to adopt the strategy.

## **References**

- [1] Journal of Biomedical Informatics, 2001. Heterogeneous database integration in biomedicine. 34(4): p. 285-98.
- [2] Sheth AP, L.J., Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput Surv, 1990. 22(3): p. 183-236.
- [3] A. Bouguettaya and R. King. Large multidatabases: Issues and directions. In IFIP DS-5 Interoperable Database Systems (Editors: D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis). Elsevier Publishers, Lorne, Victoria, Australia, 1993.
- [4] Beck, Kent; et al. (2001). "Manifesto for Agile Software Development". Agile Alliance. Retrieved 2011-10-18.
- [5] Socket Connection. <http://www.oracle.com/technetwork/java/socket-140484.html> . Retrieved 2011-10-20.
- [6] Art Taylor (Paperback - January 15, 1999). JDBC Developer's Resource (2nd Edition) .
- [7] MySQL Cluster CGE. <http://MySQL.com/products/cluster/>. Retrieved 2011-09-02.
- [8] Xiao Qin , Dynamic Load Balancing for IO-Intensive Tasks on Heterogeneous Clusters, Proceeding of the 2003 International Conference on High Performance Computing (HiPC03)
- 9] Chengzhong Xu, Francis C. M. Lau. Load balancing in parallel computers: theory and practice.
- 10] Xiao Qin, (2006). Performance comparisons of load balancing algorithms for IO intensive workloads on clusters, Journal of Network and computer applications.
- 1] Kai Lu, Riky Subrata, Albert Zomaya, "An Efficient Load Balancing Algorithm for Heterogeneous Grid Systems Considering Desirability of Grids Sites" IEEE, pp. 311-319(2006).
- 2] Replication: <http://dev.mysql.com/doc/refman/5.0/en/replication.html> . Retrieved 2012-01-20.

[13] Replication (Computing).[http://en.wikipedia.org/wiki/Replication\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Replication_(computer_science)) .Retrieved 2012-01-22.

[14] Grig Gheorghiu.Agile Testing.<http://agiletesting.blogspot.com/2009/04/mysql-load-balancing-and-read-write.html>.Retrieved 2012-01-19.

[15] Oracle and affiliate. MySQL Reference Manual ,MySQL Proxy Guide, Document generated on: 2012-02-20 (revision: 29125)

[16] Chandra Kopparapu: Load Balancing Servers, Firewalls & Caches, Wiley, ISBN 0-471-41550-2

[17] Brian Keating.Database Specialists, Inc. Challenges Involved in Multimaster Replication  
. [http://www.dbspecialists.com/files/presentations/mm\\_replication.html](http://www.dbspecialists.com/files/presentations/mm_replication.html),Retrieved 2012-02-15.

[18] MySQL Cluster Performance .[http://www.cherry.world.edoors.com/CMta\\_qxLOVfg](http://www.cherry.world.edoors.com/CMta_qxLOVfg),Retrieved 2012-03-02.

[19] Oracle and affiliate. Limitations Relating to Multiple MySQL Cluster Nodes  
. <http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster-limitations-multiple-nodes.html> .Retrieved 2012-03-02.

[20] Oracle and affiliate. Limitations Relating to Performance in MySQL Cluster.  
<http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster-limitations-performance.html> .Retrieved 2012-03-02.

[21] Oracle and affiliate. Limits and Differences of MySQL Cluster from Standard MySQL Limits  
<http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster-limitations-limits.html> .Retrieved 2012-03-02.

[22] Database Expertise.<http://pythian.com/news/896/simple-mysql-proxy-failover> .Retrieved 2012-03-05.

[23] Selecting a Platform for your MySQL server [http://dev.mysql.com/techresources/articles/mysql\\_platform\\_selection.html](http://dev.mysql.com/techresources/articles/mysql_platform_selection.html) .Retrieved 2012-03-05.

# Appendix 1: Sample CPU Usage Code

```
package server;

import java.lang.management.*;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.atomic.AtomicLong;

public class MultiCoreTester {
    private static final int THREADS = 5;
    private static CountDownLatch ct = new CountDownLatch(THREADS);
    private static AtomicLong total = new AtomicLong();

    public double PrintExecutionTime() throws InterruptedException{
        System.out.println("*****CPU RESOURCES*****");

        long elapsedTime = System.nanoTime();
        for (int i = 0; i < THREADS; i++) {
            Thread thread = new Thread() {
                public void run() {
                    total.addAndGet(measureThreadCpuTime());
                    ct.countDown();
                }
            };
            thread.start();
        }
        ct.await();
        elapsedTime = System.nanoTime() - elapsedTime;

        System.out.println("Total elapsed time " + elapsedTime);
        System.out.println("Total thread CPU time " + total.get());
        double factor = total.get();
        factor /= elapsedTime;

        //System.out.println("CPU time to elapsed time factor is " + factor);
        System.out.println("Factor: %.2f%n " + factor );

        System.out.println("-----");
        return factor;
    }

    private static long measureThreadCpuTime() {
        ThreadMXBean tm = ManagementFactory.getThreadMXBean();
        long cpuTime = tm.getCurrentThreadCpuTime();

        long total=0;
        for (int i = 0; i < 1000 * 1000 *100 ; i++) {
            // Keep the loop busy for a while to collect information
            total += i;
        }
    }
}
```



```
total *= 10;
}
```

```
cpuTime = tm.getCurrentThreadCpuTime() - cpuTime;
System.out.println(total + " ... " + Thread.currentThread() +
    ": cpuTime = " + cpuTime);
```

```
return cpuTime;
}
```

```
public double PrintCPULoad() throws InterruptedException {
    OperatingSystemMXBean mxbean_ = ManagementFactory.getOperatingSystemMXBean();

    System.out.println("My processors " + mxbean_.getAvailableProcessors());

    System.out.println("My Load " + mxbean_.getSystemLoadAverage());

    return mxbean_.getSystemLoadAverage();
}
}
```

## Appendix 2: Sample Memory Usage Code

```
package server;
```

```
public class MemoryUsage {
    public double printUsage(Runtime runtime) {
        long total, free, used;

        byte[] bytes;

        // Print initial memory usage.
        runtime = Runtime.getRuntime();

        // Allocate a 1 Megabyte and print memory usage
        bytes = new byte[1024*1024];

        bytes = null;
        // Invoke garbage collector to reclaim the allocated memory.
        runtime.gc();

        // Wait 5 seconds to give garbage collector a chance to run
        // try {
        //     Thread.sleep(5000);
        // } catch (InterruptedException e) {
        //     e.printStackTrace();
        // }
```

```

// }

total = runtime.totalMemory();
free = runtime.freeMemory();
used = total - free;

System.out.println("*****MEMORY RESOURCES*****");

System.out.println("\nTotal Memory: " + total);
System.out.println("    Used: " + used);
System.out.println("    Free: " + free);
System.out.println("Percent Used: " + ((double)used/(double)total)*100);
System.out.println("Percent Free: " + ((double)free/(double)total)*100);

System.out.println("-----");
System.out.println(" ");
return ((double)used/(double)total)*100;
}

}

```

### Appendix 3: Sample Server Code

```

package server;
import client.*;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.lang.Runnable;
import java.lang.Thread;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

import org.omg.CORBA.PRIVATE_MEMBER;

public class ServerSocketConnect {

private ServerSocket server;

private int port = 7777;

public ServerSocketConnect() {

try {

server = new ServerSocket(port);

} catch (IOException e) {

e.printStackTrace();

```

```
}  
  
}  
  
public static void main(String[] args) {  
    ServerSocketConnect example = new ServerSocketConnect();  
    example.handleConnection();  
}
```

```
public void handleConnection() {  
    System.out.println("Waiting for client request.....");  
  
    // The server do a loop here to accept all connection initiated by the  
    // client application.  
    //  
    while (true) {  
        try {  
            Socket socket = server.accept();  
            new ConnectionHandler(socket);  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
class ConnectionHandler implements Runnable{
```

```

private Socket socket;

public ConnectionHandler(Socket socket) {

this.socket = socket;

Thread t = new Thread(this);

t.start();

}

public void run() {

try

{

//

// Read a message sent by client application

ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());

String message = (String) ois.readObject();

System.out.println("Clients Message: " + message);

System.out.println(" ");

// Send a response information to the client application

ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());

oos.writeObject(ArrayParam());

oos.flush();

ois.close();

oos.close();

socket.close();

System.out.println("Waiting for client message...");

} catch (IOException e) {

```

```

e.printStackTrace();
} catch (ClassNotFoundException e) {
e.printStackTrace();
} catch (InterruptedException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

public ArrayList<Double> ArrayParam() throws InterruptedException{
    ArrayList<Double> al=new ArrayList<Double>();

    MemoryUsage mu=new MemoryUsage();
    Runtime runtime;
    byte[] bytes;

    // Print initial memory usage.
    runtime = Runtime.getRuntime();

    double MemUsage=mu.printUsage(runtime);

    //-----CPU usage
    double CpuUsage=0;
    MultiCoreTester mct=new MultiCoreTester();
    CpuUsage = mct.PrintExecutionTime();

    al.add(MemUsage);
    al.add(CpuUsage);
    al.add(mct.PrintCPULoad());
    return al;
}
}

```

## Appendix 2: Sample Client Code

```

package client;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.ObjectInput;

```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.ClassNotFoundException;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.sql.*;
```

```
import java.sql.*;
```

```
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.*;
import java.io.*;
```

```
public class ClientSocketConnect {
    static double LargestResource=0;

    public static double ClientConnect(String host) {
        double avgResults=0;
        try {
            //
            // Create a connection to the server socket on the server application
            //
            // String host = "196.168.1.189";
            Socket socket = new Socket(host, 7777);

            //GFG
            // Send a message to the client application
            //
            long InitialTime=System.nanoTime();

            double AvgTime=0.0;
```

```

System.out.println(" ");
System.out.println("-----"+ host + "-----");

ObjectOutputStream oos = new ObjectOutputStream(socket.getOutputStream());
oos.writeObject("Confirmation : Client Sent a request");

//
// Read and display the response message sent by server application
//
ObjectInputStream ois = new ObjectInputStream(socket.getInputStream());
//String message = (String) ois.readObject();
ArrayList<Double> al= (ArrayList<Double>) ois.readObject();

System.out.println("Confirmation: "+ host +" Server returned the message with items");

double elapsedTime=(System.nanoTime() -InitialTime)/1000000000.0;

//
AvgTime=((InitialTime/elapsedTime) % 100) ;

al.add(elapsedTime);

Object[] ia=al.toArray();

for (int i = 0; i < ia.length; i++) {
    System.out.println(i+1 + " : " + ia[i]);
    avgResults+= Double.valueOf(ia[i].toString()).doubleValue() ;
}

ois.close();
oos.close();

} catch (UnknownHostException e) {
    e.printStackTrace();
}

```

```

} catch (IOException e) {
    //e.printStackTrace();
    System.out.println(" ");

    System.out.println("XXXXXXXXXX " + host + " is unreachable XXXXXXXXXX");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
    return avgResults;
}

```

```

public static void main(String[] args){

```

```

    int sCounter=1;
    String[] ipArray = null;
    ipArray = new String[0];

    try{
        FileInputStream fstream = new FileInputStream("/home/xb/ips");
        // Get the object of DataInputStream
        DataInputStream in = new DataInputStream(fstream);
        BufferedReader br = new BufferedReader(new InputStreamReader(in));
        String strLine;
        //Read File Line By Line
        while ((strLine = br.readLine()) != null) {

            ipArray=(String[]) resizeArray(ipArray,sCounter);
            ipArray[sCounter-1]=strLine;

            sCounter+=1;

        }
        //Close the input stream
        in.close();
    } catch (Exception e){//Catch exception if any

```



```
//e.printStackTrace();
```

```
}
```

```
boolean isReachable=false;
```

```
String LargestIP=null;
```

```
double[] ServerArray=new double[ipArray.length];
```

```
double clients=0;
```

```
double largest=0;
```

```
for (int i = 0; i < ipArray.length; i++) {
```

```
    clients= ClientConnect(ipArray[i]);
```

```
    //System.out.println (ipArray[i]);
```

```
    ServerArray[i]=clients;
```

```
    if(i==0){
```

```
        largest=clients;
```

```
        LargestIP=ipArray[i];
```

```
    }
```

```
    if((largest > clients) && (clients !=0) )
```

```
    {
```

```
        largest = clients;
```

```
        LargestIP=ipArray[i];
```

```
    }
```

```
    if (clients>0) {
```

```
        System.out.println(" ");
```

```
        System.out.println("Resources usage for " + ipArray[i] + " is " + clients );
```

```
    }
```

```
}
```

```

System.out.println(" ");
System.out.println("*****");

if (largest<=0) {
    System.out.println("All the servers are down");
    System.exit(-1);

    }else{
        System.out.println("The best sever is : " + LargestIP + " " + largest);
        dbConnect(LargestIP);
    }

}

```

```

private static void dbConnect(String host){
    Connection conn = null;
    long DBstartQueryTime=System.nanoTime();

    System.out.println (" ");
    System.out.println ("Attempting DB Connection.....");
    try
    {
        String userName = "root";
        String password = "root";
        String url = "jdbc:mysql://" + host + "/meo";
        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        conn = DriverManager.getConnection (url, userName, password);
        System.out.println ("Database connection established");

        Statement s = conn.createStatement ();
        s.executeQuery ("SELECT * FROM v_memberstatement");
        ResultSet rs = s.getResultSet ();
        int count = 0;
    }
}

```

```

//. System.out.println ("Payroll No | Activity Ref");
while (rs.next ())
{

String sName = rs.getString (2);
String sTown = rs.getString (3);

//System.out.println (sName + " | " + sTown);

++count;
}

double DbElapsedTime= (System.nanoTime()-DBstartQueryTime)/1000000000.0;

System.out.println("DB Query Time was : " + DbElapsedTime + " Sec's");

rs.close ();
s.close ();
System.out.println (count + " rows were retrieved");
}
catch (Exception e)
{
e.printStackTrace();
System.err.println ("Cannot connect to database server");
}
finally
{
if (conn != null)
{
try
{
conn.close ();
System.out.println ("Database connection terminated");
}
catch (Exception e) { /* ignore close errors */ }
}
}

```

```
}  
  
private static Object resizeArray (Object oldArray, int newSize) {  
    int oldSize = java.lang.reflect.Array.getLength(oldArray);  
    Class elementType = oldArray.getClass().getComponentType();  
    Object newArray = java.lang.reflect.Array.newInstance(  
        elementType,newSize);  
    int preserveLength = Math.min(oldSize,newSize);  
    if (preserveLength > 0)  
        System.arraycopy (oldArray,0,newArray,0,preserveLength);  
    return newArray;  
}
```

