Automated Control of Elasticity for a Cloud-Based Key-Value Store

ALA ARMAN

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2012

TRITA-ICT-EX-2012:20

ROYAL INSTITUTE
OF TECHNOLOGY

Automated Control of Elasticity for a Cloud-Based Key-Value Store

Master of Science Thesis

Ala Arman

Examiner:

Vladimir Vlassov

Supervisor:

Ahmad Al-Shishtawy - PhD Candidate

Kungliga Tekniska Högskolan (KTH)

Stockholm 2011

Abstract

“Pay-as-you-go” is one of the basic properties of Cloud computing. It means that people pay for the resources or services that they use. Moreover, the concept of load balancing has been a controversial issue in recent years. It is a method that is used to split a task to some smaller tasks and allocate them fairly to different resources resulting in a better performance. Considering these two concepts, the idea of “Elasticity” comes to attention. An Elastic system is one which adds or releases the resources based on the changes of the system variables. In this thesis, we extended a distributed storage called Voldemort by adding a controller to provide elasticity. Control theory was used to design this controller. In addition, we used Yahoo! Cloud Service Benchmark (YCSB) which is an open source framework that can be used to provide several load scenarios, as well as evaluating the controller. Automatic control is accomplished by adding or removing nodes in Voldemort by considering changes in the system such as the average service time in our case. We will show that when the service time increases due to increasing the load, as generated by YCSB tool, the controller senses this change and adds appropriate number of nodes to the storage. The number of nodes added is based on the controller parameters to decrease the service time and meet Service Level Objectives (SLO). Similarly, when the average service time decreases, the controller removes some nodes to reduce the cost of using the resources and meet “pay-as-you-go” property.

Keywords: Cloud Computing, Elastic Computing, Control Theory, Voldemort, YCSB, Key-Value Store

Acknowledgment

First and foremost, I wish to thank Prof. Vladimir Vlassov who gave me the honor to work with him during my Master Thesis. This provided me a nice working atmosphere together with great recommendations and solutions.

Next, I want to express my gratitude to my supervisor, Mr. Ahmad Al-Shishtawy, which this thesis would not have been possible unless his cooperation. I gained amazing experiences and expertise under his supervision.

I also would like to make a special reference to Mr. Roshan Sumbaly who is one of the main committers of the Voldemort. Without his corporation I could not have progressed during the project.

I offer my regards and blessings to all of my friends who supported me in any respect during the completion of the project.

Finally, I thank my family for moral supporting me throughout all my studies at KTH University.

Table of Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

Cloud computing is in high demand by various IT organizations. It is an infrastructure that provides resources, information and services as a utility and not as a product over the Internet [1]. There are two main advantages with Cloud computing. The first one is that the end user does not need to be involved in the configuration of the system. The second one and probably the most important, is that the users only pay for a service whenever they use it. That is why the Cloud computing approach is less expensive than their software alternatives. However, this property, called “pay as you go” has an important drawback. If allocated resources exceed the required amount, it will incur an additional fee. In addition, if the available resources are not adequate, the system cannot meet the SLO, resulting in a negative impact on its performance.

When the concept of Cloud computing emerged in the IT arena, there have been always challenges to have the flexibility to meet business needs and workloads. For example, the behavior of user’s use the resources change during night and day or during the growing phase of a social network website, and as a result more computational resources are needed to handle the workload [2]. These are the issues that Elastic computing deals with in a manner such that the system is able to add or remove resources based on changes within itself.

We will design and implement an automatic controller which utilizes control theory considering the elasticity property in a distributed storage called Voldemort. In other words, the controller would be an extension to Voldemort in a way that it scales up by allocating more nodes in the case of a high load and scales down by removing some nodes in the case of a decreasing load, based on a predefined algorithm. The goal here is that a system that would use the resources in an efficient way, so as not to waste resources in the case of a light load. In addition, it adds some nodes to meet SLO in case of increasing workload. Moreover, the automatic controller

eliminates the need for the administrator of the system to configure the system manually to meet the first main advantage of Cloud computing.

In this thesis we will focus on the storage layer of the web 2.0 three tire applications. We will use a distributed storage called Voldemort which is currently being used in LinkedIn. Moreover, we will use YCSB, an open source framework, to simulate changing the behavior of end users on the storage.

1.1. Thesis Objectives

In this section we shall present the several objectives of the thesis. We covered several concepts that were related to this research area as well as extension of Voldemort such that it could handle the elasticity of resources in an automatic way. The major objectives of this thesis are as following:

- **A survey about Cloud computing**: Cloud computing is one the most important concepts in recent years in the IT world. It is becoming more and more widely used due to its unique properties such as lowered cost, less maintenance, easy configuration, fewer updating problems, and easy access to information, resources or services. Some details about Cloud concepts, the properties of Cloud computing and other important concepts in this area should be covered.
- **A survey about Distributed Storage Systems (DSSs):** DSSs currently constitute a significant part of storage systems. One of the most important properties of these storages is their scalability. However, there are some issues such as reliability and consistency. As a real distributed storage is used in the thesis, it is important to know about the design and architecture of some DSSs, reasons for choosing one of them for our project.
- **Studying Voldemort:** The design, architecture and configuration and other aspects of Voldemort such as failure detector and rebalancing tool[1] should be discussed.
- **Studying YCSB:** It is a free framework to benchmark which provides different workload scenarios by creating some threads which act as clients and express the results. We used it as a workload generator. The architecture of YCSB and its usage to generate some workload schemes for Voldemort should be discussed.
- **Studying Control Theory and its application in Cloud computing (Elastic computing):** Control theory has been used mostly in electrical and mechanical engineering, but it has also been applied to computing systems. The use of control theory to design an Autonomic controller to handle the elasticity of the resources in Voldemort should be presented.
- **Design method of the controlling framework:** A framework which handles the resource controlling in a way that the system would scale up or down in cases of an increase or a decrease in the workload, respectively should be introduced. System

[1] A tool that adds or removes nodes in Voldemort.

identification, designing the model of the system and designing the controller should be done using Matlab.

- **Extending Voldemort to implement the Elastic controller:** The controller should be added to Voldemort in a way such that it could handle the load change by adding or removing nodes. The solution should be implemented such that the controller would connect to each node in the cluster and take the average service time from them at the end of a defined interval. Based on these statistics decide, it would decide on the number of nodes that should be added or removed.
- **Experiment using the controller and Evaluation of results:** The controller should be tested to find out how it operates under experimental conditions using parameters that have been specified in the designing process.

1.2. Thesis outline

This report is going to be in a number of chapters described below:

- **Chapter 1**: This chapter includes a general definition to the problem, the thesis objective and the thesis outline.
- **Chapter 2**: This chapter begins with an introduction about the Cloud computing. Then it continues with the studying of some distributed storage systems including Voldemort. Elastic computing and Autonomic computing and their different aspects are then discussed, as well as discussing YCSB as a benchmark framework and system identification and control theory. The chapter is finalized with the presenting of two important related works.
- **Chapter 3**: This chapter begins with introducing the controlling framework architecture in detail and its different components, their design and implementation. Then we continue with, system identification and system modeling and controller design.
- **Chapter 4**: This chapter begins with the continuation of identification of the system. Experiments using the controller are then discussed and evaluated.
- **Chapter 5**: This chapter includes the summary of what we have done during the master thesis, conclusions, and introducing some ideas for future work.

Chapter 2

2. Background

In the previous chapter, we discussed some basic concepts that are related to our project together with thesis objective and thesis outline. In this chapter, we will talk more about these important concepts that a reader should know for a better understanding. Then we will introduce some related works that has been done before.

2.1. Cloud Computing

Cloud computing is an expression that has become controversial these days. We first focus on the concept of Cloud in the computing field and then go through some discussions about Cloud computing and its different aspects.

2.1.1. Cloud

As was mentioned in chapter 1, in the context of software and Internet, Cloud is a metaphor for describing Internet that includes online resources that permits users to access data and applications from any device. End users should not have to deal with the configuration of the software and hardware services that they are going to use them [3]. For Example when we use electricity, we do not need to deal with turbine, generator or magnets. Therefore, in the context of electricity turbine, generators or magnets are pieces in the Cloud that is the electricity company.

To apply this in the context of the computing area, we can say that we do not have to deal with the server configuration, system maintenance, updating issues etc... We just plug in using the Internet and use the power of them.

2.1.2. Cloud computing features

According to [3], “Cloud computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services)” . As a result it is a new perspective that enables a new platform and location independent to how the end users communicate. Figure (1), represents the idea of Cloud computing.

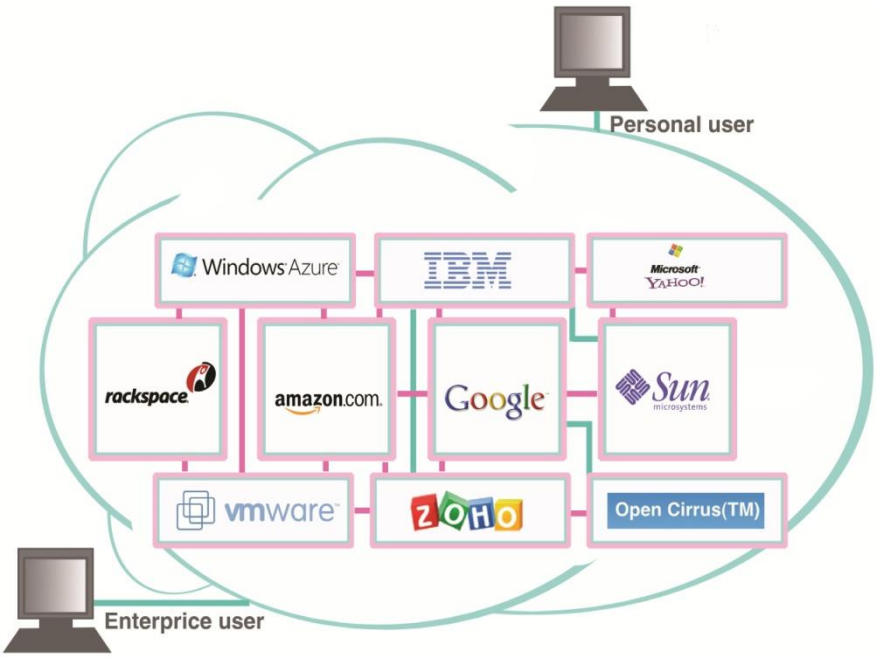

Figure 1 : Schematic representation of CLOUD computing (Adopting from Figure 1 of [4])

Cloud computing is closely associated with web 2.0. Here we will mention three important concepts in the Cloud computing:

Software as a Service (SaaS): Figure (2), shows the idea of SaaS. In this type of service, the customer can use the application without dealing with OS, hardware, update issues, network configuration etc… in another word, it enables users to run their application in a Cloud environment. In fact the consumer does not have control on them [3].

- **Commercial online management tools:** Clarizen

- **Customer relationship management and Human Resource Management:** Salesforce.com, Zoho
- **Online Office applications:** GoogleDoc, SlideRocket

- **Platform as a Service (PaaS):** Service here is a hosting environment that user controls his running application in this environment such as E-Governance applications like G2B[2], G2C[3], G2E[4] [3].
- **Infrastructure as a Service (IaaS):** In this type of services, the end user runs his application on a Cloud environment and also can have some control on some parameters in the Cloud such as storage, processing, firewalls etc… [3].

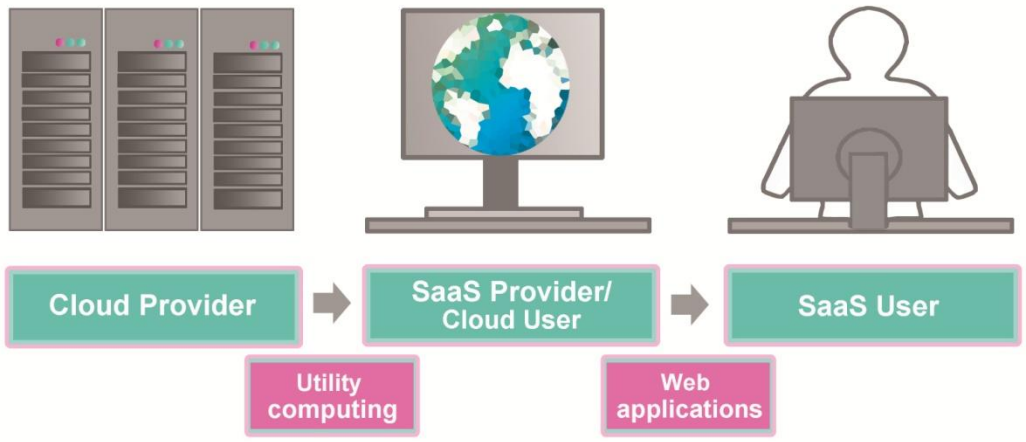

Figure 2 : Users and providers of Cloud computing In SaaS. Adopted from Figure 1 of [5])

Cloud computing has too many advantages in comparison with local applications; some of them are as following:

- **Scalability**: Users just connect to the online application thorough the Internet and there is no worry about the number of the users. You can add or subtract the capacity as you need.
- **Real Time, Automated Upgrades**: The software is updated once and users all over the world can see it in the same time.
- **Faster way to build applications**: Developer does not have to buy any servers, security providers, data centers etc... . Only the developer builds his applications and he is done.
- **Less cost**: The end users only pay for the resources when they use them. It reduces the using costs significantly.
- **Easy to implement:** The end users do not have to worry about software licenses or the implementation of the services. [3]

[2] Government to Business
[3] Government to Consumer
[4] Government to Enterprise

2.2. Distributed (Storage) Systems

A Distributed system, according to [6], is one in which failing a computer that the user does not know about its existence in the system, can render his system useless. As distributed systems are built up on the top of networks and try to provide the user a single entity with whatever services that is required. The main featurs of distributed systems are [7]:

- **Functional Separation**: The functional role of each entity in the system is separated based on their capability and purpose.
- **Inherent distribution:** The information of distributed entities such as people and system could be generated, stored and analyzed.
- **Reliability**: It should keep data safely for long term (backup and replication).
- **Scalability**: It should be able to add resources to increase the performance and availability of the system.
- **Economy**: It should share the resources by many entities to reduce the cost.

There are various types of distributed systems such as peer-to-peer systems, Grid systems, cluster and distributed storage systems. As we used a distributed storage called Voldemort in our thesis, the concept of distributed storages and some examples are discusses in the following:

2.2.1. Distributed Storage Systems (DSSs)

Distributed Storage Systems, provide users a system in a way that they have a unified view of stored data on different file systems and computers [7]. They try to keep data available by a concept called redundancy. It means that if a node fails, new redundant fragments are introduced to the system based on the level of the availability that is defined for the system [8]. There have been several types of distributed storages introduced. In the next section we go through some of them that are more famous:

2.2.1.1. Cassandra

Cassandra [9] is a distributed storage system which is used in Facebook and was open sourced by it in 2008. It is designed to manage highly structured data spread out across many commodity servers and support performance quality, reliability, efficiency, and continuous growth in Facebook. Other highlights of Cassandra are [10]:

- Eventual consistency
- Tunable tradeoffs between consistency and latency
- Minimal administration
- No SPF (Single Point of Failure)

- **Data model**: Cassandra is a table based distributed storage. A table is a map with multi dimensions which is distributed and indexed by a key.
  Architecture: Like all DHTs, it consists of a ring with some nodes with 128 bits namespaces that provides a good distribution for keys. Every node picks a random identifier in namespace through a hash function. Each item <Key, Value> gets an identifier H (Key) = k. And finally each item is stored at its successor.

2.2.1.1.1. Scalability

In this section we shall present how Cassandra adds or removes nodes in a cluster automatically. To do this, *nodetool* utility should be used. It provides a command line interface. In the following table we can see the parameters of this utility that should be applied [10]:

Table 1: Parameters to for *nodetool* utility in Cassandra

Parameter	Description
host	The name or IP of the adding/removing node
port	The port for connecting to the node
password	The password for connecting to the node
username	The username for connecting to the node
join	Join the node to the ring (cluster)
decommission	Take the node out of the ring (remove the node)
Load balance	Load balance the node

There are some advantages to use this utility in Cassandra; some of them are as following:

- No down-time or interruptions
- No need to reconfigure or reboot the cluster(s).

There are some disadvantages to use this utility to scale nodes in Cassandra; some of them are as following:

- Data is not removed from the decommission node automatically.
- Adding node is a slow process if each node is responsible for large amount of data.
- Currently, this utility should be run in the same environment as Cassandra as well as the same classpath.

2.2.1.2. Hbase

Hbase [11], is an open source, distributed column oriented store that is built on top of the Hadoop[5] which has been designed and implemented by Apache.

- **Data Model**: Data is stored in the tables which are made of rows and columns. All columns belong to a column family. Tables consist of some version cells that their contents are un-interpreted arrays of bytes. All tables are accessed via their primary key which is row key. Table (2) shows a table in Hbase.

Table 2 : A table in Hbase (Adopted from [11])

Row Key	Time Stamp	Column Family Contents	Column Family anchor
"com.cnn.www"	t9		Anchor:cnnsi.com="CNN"
	t8		Anchor:my.look.ca="CNN"
	t6	Contents:html="<html>…"	
	t5	Contents:html="<html>…"	
	t3	Contents:html="<html>…"	

- **Architecture** : Hbase has 4 major components[34] :
  - **HbaseMaster**: Assigns regions to HRegionServers. The first assigned region is ROOT region, which locates all META regions to be assigned. Each META region maps a number of user regions.
  - **HregionServer**: that manages client request (reading and writing)
  - **Hbase Client**: locates particular HregionServers and location of user region. Then client contacts the related region server serving a particular region and provides read and write requests.

2.2.1.2.1. Scalability

In this section we shall discuss how Hbase handles the scalability issue in a practice. This distributed storage system, can scale up and down horizontally. There are different approaches for stopping region servers and restarting them as following [11]:

- **Stopping a region server:** A region server can be stopped by running the following script:

./bin/graceful stop.sh HOSTNAME

The process of the stopping a regions server includes following steps:

- All the regions turn off once at a time.
- The region server with *hostname= HOSTNAME* is stopped.

[5] Hadoop is a framework designed by apache that "allows for the distributed processing of large data sets across clusters of computers using a simple programming model"

- All other regions are redeployed.

- **Restarting a regions server**: For restarting a regions server, the following steps should be done:

  - Ensuring the cluster consistency (by running *./bin/hbase hbck*)
  - Restart the master server (by running *./bin/hbase-daemon.sh stop master; ./bin/hbase-daemon.sh start master*)
  - Disable region load balancer (by running *echo "balance_switch false" | ./bin/hbase shell*)
  - Run *graceful_stop.sh script as following:*

    ./bin/graceful_stop.sh --restart --reload --debug HOSTNAME

    It will move back the offloaded region to the server.

  - Restart the Master server again.
  - Run hbase to ensure the consistency of the cluster.

There are some disadvantages of stopping or rebalancing the region servers in Hbase. Some of them are as following:

- ✓ **Down Times**: Before stopping the region servers, all regions should be turned off. Similarly, in restarting a region server, the master server should be restarted before and after restarting the region server.
- ✓ **Manual process**: As was mentioned, restarting a region server includes some processes that should be done manually.
- ✓ **Manual load balancer**: The load balancer during the stopping or restarting a region server should be disabled; otherwise, there might be a contention between restarting the master server and the load balancer.
- ✓ **Inability to use IP**: According to [11], at the time of writing this thesis, the *graceful_stop.sh* script is not smart enough to take the hostname out of IP.

2.2.1.3. MongoDB

MongoDB is “a scalable, high-performance, open source, document-oriented database, Written in C++.” [35]

- **Data model:** In MongoDB data model has 2 important properties [36]:

  - **Document oriented**: As was mentioned earlier, MongoDB is a document oriented not relational. It means that the concept of a “row” is replaces by a concept called document. By exploiting this approach, handling complex hierarchical relationships with a single record is possible.

- **Schema-Free:** A document's keys are not predefined or fixed. This property causes that massive data migrations get usually unnecessary.

- **Architecture:** mongoDB has 2 major components(processes) to the database server [35]:

  - **Mongod**: it is the core database server. Most of the times, mongod could be used as a self-contained system like mysqld on a server.
  - **Mongos**: It provides auto-sharding[6]. It can be considered as a "database router" to make a cluster of mongod processes as if they are a single database.

2.2.1.3.1. Scalability

In the following we will discuss about the scalability issue in MongoDB. In this distributed storage, the shards can be added to the cluster or removed from it. A utility called Sharding tool is used to achieve this goal. The processes of scaling up or down are as following:

- **Adding a shard:** Using the sharding tool, a new shard can be added to the cluster's configuration using following command:

```
db.runCommand ({addshard: "<serverhostname> [:< port>]"});
```

- **Removing a shard:** Each shard has a name. Using the sharding tool, a shard can be removed from the cluster by using following command:

```
db.runCommand ({"removeshard: "shard0000"});
```

There are some advantages of adding or removing shards in MongoDB. Some of them are as following:

- **No down Time**: Shards can be added or removed without any down time in the cluster.
- **Automatic load balance**: the load of adding or removing shards is balanced automatically during adding/removing shards.

2.2.1.4. Voldemort

Voldemort is a distributed, persistent fault-tolerant non-relational key-value hash table. It is used at LinkedIn for certain high-scalability storage problems where simple functional partitioning is not sufficient. The main characteristics of Voldemort are as following:

[6] In mongoDB, shard refers to one or more servers and stores data using mongoDB processes. Auto-sharding provides partitioning, automatic balancing for changes in load and data distribution, adding new machines easily and failover automaticity in mongoDB.

- **Data Replication**: Data is replicated over multiple servers automatically.
- **Data Partitioning:** Data is automatically partitioned in a way that each server contains only a portion of the total data.
- **Data Versioning**: Data items are versioned to maximize data integrity in failure scenarios without compromising availability of the system
- **Node independency:** Each node is independent of other nodes with no central point of failure or coordination.
- **(Good) single node performance**: 10-20k operations/second depending on the machines, the network, the disk system, and the data replication factor.

2.2.2. Voldemort Design

As was mentioned above, Voldemort is a key-value storage in which values and keys can be complex compound objects including maps or lists that enables high performance and availability. In Voldemort there is a concept called “store” that is equivalent to table in relational databases. Each key is unique to a store and each key can have at most one value. In the next section we will present the architecture of the Voldemort:

2.2.2.1. Architecture

As we can see in Figure (3), the architecture of Voldemort consists of several layers that are simple storage interfaces for *put*, *get* and *delete*. Each layer performs one action such as TCP/IP network communication, serialization, version reconciliation, inter-node routing. For example, routing layer is responsible to take an operation and delegating it to all *n* storage nodes.

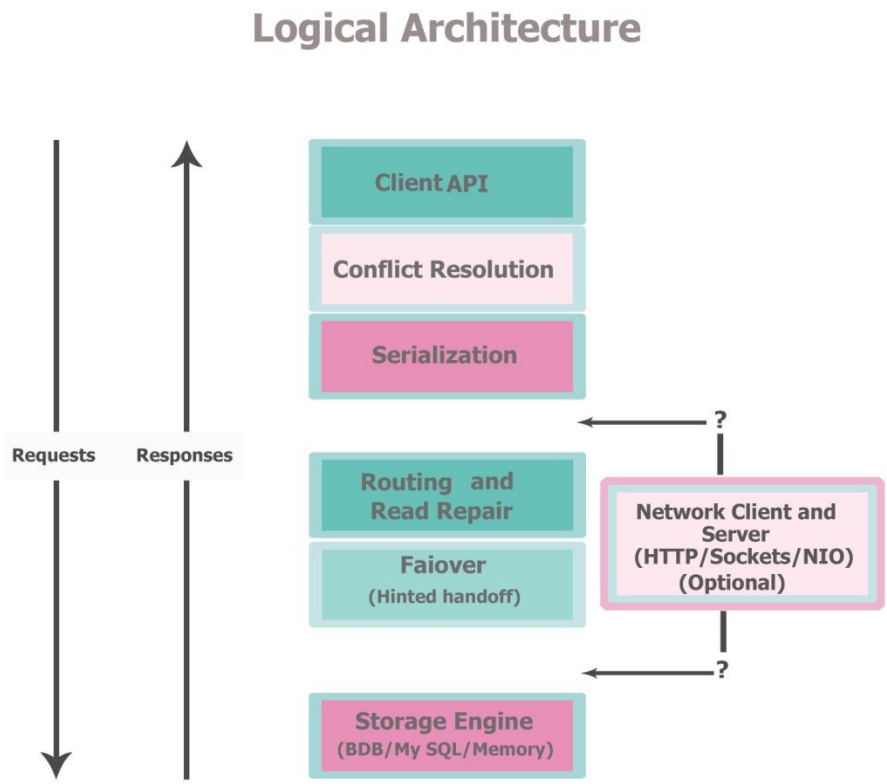

Figure 3 : Voldemort Architecture. (Adopted from[37])

2.2.2.2. Data Partitioning

Data in Voldemort is partitioned among the cluster of servers. Data partitioning has two major advantages:

- Even data can fit on the disk, no single node needs to keep the whole data. Because disk access for small values is dominated by seeking time. Therefore, partitioning improves the cache efficiency. It is done by splitting the set of data to the smaller chuncks that can be stored on the memory of the server that contains that partition. In other words, the nodes in the cluster are not interchangeable and each request should be routed to the node that holds that partition.
- Because of the existence of node failure due to maintenance, or becoming overloaded, if we assume that there are *s* servers in the cluster and each of them fail with the probability of *p* in a day, then probability of loosing at least one server in a day is 1- $(1-p)^n$, Considering that storing all data on one server is not reasonable.

The approach of Voldemort to accomplish effective data partitioning is somehow simple in a way that data is cut to *s* partitions and copies of the given keys are stored in the *r* other nodes if replication degree is *r*. To associate r servers with key k, taking a = *k* mod s and storing values

in a, a+1, …, a+r. By using this approach, the location of a key can be easily computed without asking central metadata server.

2.2.2.3. Consistent Hashing

Consistent hashing is a technique that is used in Voldemort to calculate the location of each key on the cluster. The advantage of using this approach is that load is equally distributed among all remaining nodes when a node is removed. Similarly, when a new node is added, only 1/*s*+1 must be moved to the new node.

To visualize this concept, we can consider some hash values as a ring beginning from 0 to 2^31 -1. This ring is divided to equally–sized partitions which Q>>S. Therefore each server is mapped to Q/S of partitions. A key is mapped to the ring by using an arbitrary hash function; Then the R replica nodes is calculated by choosing first R unique nodes when moving over partitions clockwise. Figure (4), shows a hash ring when S=4 and R=3.

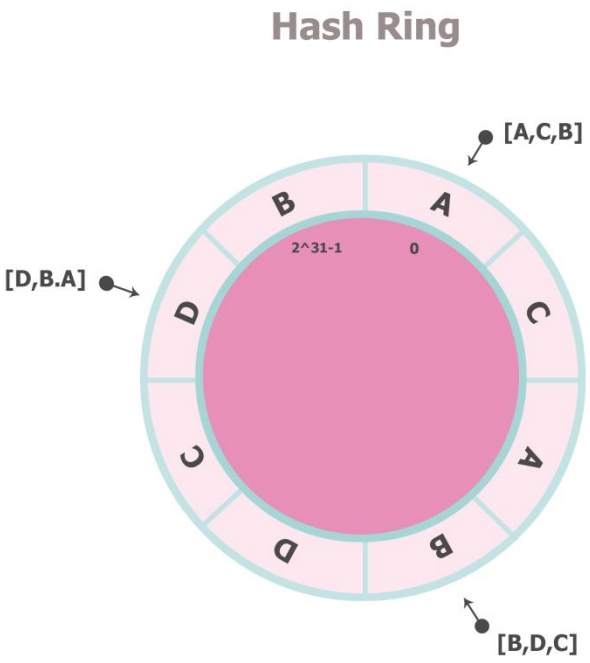

Figure 4 : A hash Ring in Voldemort. (Adopted from [37])

2.2.2.4. Data model and serialization

Serialization in Voldemort is pluggable in a way that either one of the ready serializes can be used or a new one can be written. The lowest level of data format in Voldemort is just arrays of bytes for both keys and values. Higher levels can be configured easily. In Table (3), the list of supported types is presented.

Table 3 : Supported data types in Voldemort. (Adopted From [37])

Type	storable sub-styles	bytes used	Java type	example type definition
number	int8, int16, int32, int64, float32, float64, date	8, 16, 32, 64, 32, 64, 32	Byte, Short, Integer, Long Float, Double, Date	"int32"
string	string, bytes	2 + length of string or bytes	String, byte[]	"string"
boolean	boolean	1	Boolean	“boolean”
object	object	1 + size of contents	Map<String,Object>	{"name":"string", "height":"int16"}
array	array	size * sizeof(type)	List<?>	["int32"]

2.2.2.5. Consistency and Versioning

In this section we first discuss consistency approaches in Voldemort and then Versioning in distributed storage is explained:

- **Consistency:** Voldemort uses 2 approaches to reach consistency:
  - **Read-Repair and Versioning**: In Read-Time conflicts are detected and all inconsistent versions are updated which involves some co-ordination and is completely fault-tolerant. It brings the best availability and highest efficiency.
  - **Hinted-Handoff**: if a node failure is detected, a “hint” of updated value is stored on one of the nodes which are alive. When the failed node came back, the updated values are copied to.
- **Versioning** In Voldemort, Versioning includes vector clock and partial order techniques. In Vector clock approach, clock keeps a counter of each writing server and allows us to compute when two versions are in conflict; and one version succeeds and or precedes another. In the following we can see an example of a vector clock that is a list of *Server: Version* :

  [1:23, 2:3, 4:42]

  “A version v1 succeeds a version v2 if for all i, v1i > v2i. If neither v1 > v2 nor v1 < v2, then v1 and v2 co-occur, and are in conflict.”For example:

  [1:2, 2:1]
  [1:1, 2:2]

are two conflicting versions which imply partial order approach.

2.2.3. Voldemort Configuration

The config directory in Voldemort project contains almost all setting parameters for the storage nodes. In the next sections the details of configuration for the Voldemort are presented:

2.2.3.1. cluster.xml

Voldemort includes some cluster of nodes. Each node is independent of other nodes with no central point of failure or coordination. The information about the cluster is kept in the *cluster.xml* file which is located in *config* directory. This file is identical for all nodes. Figure (5) shows a typical cluster.xml file in Voldemort.

```
<cluster>
    <name>mycluster</name>
    <server>
        <id>0</id>
        <host>s1802.it.kth.se</host>
        <http-port>8081</http-port>
        <socket-port>6666</socket-port>
        <partitions>0, 1</partitions>
    </server>
    <server>
        <id>1</id>
        <host>s1803.it.kth.se</host>
        <http-port>8082</http-port>
        <socket-port>6667</socket-port>
        <partitions>2, 3</partitions>
    </server>
</cluster>
```

Figure 5 : An example for a Cluster.xml file in Voldemort

As we can see, in this cluster with the name of “mycluster”, we have two nodes that are on machines named s1802.it.kth.se and s1803.it.kth.se. We also can define the http port and socket port for each nodes. The important point here is that each node can have partitions from 0 to 2^31-1. In the above example we can see that we have 4 partitions that partitions 0 and 1 are on the host with id=0 and partitions 2 and 3 are one node with id=1. Partitions should start from zero.

2.2.3.2. stores.xml

Stores in the Voldemort are the same concept as tables in relational databases. We can have several stores in a cluster. The information about the stores in the cluster is kept in a file named *stores.xml in config* directory. Figure (6) shows a typical stores.xml:

```
<store>
  <name>test</name>
  <persistence>bdb</persistence>
  <routing>client</routing>
  <replication-factor>2</replication-factor>
  <required-reads>2</required-reads>
  <required-writes>2</required-writes>
  <key-serializer>
    <type>string</type>
  </key-serializer>
  <value-serializer>
    <type>string</type>
  </value-serializer>
  <retention-days>1</retention-days>
</store>
</stores>
```

Figure 6 : An example for a stores.xml in Voldemort

As we can see, this file shows that in the related cluster, there is only one store named “test”. Persistence is bdb (Berkely DB), means the storage engine which is used in the persistence layer is Berkely DB. The routing type is “client” (side), replication factor is 2, required writes and required reads are 2. Moreover, we have serializes for key and values that both of them are string. And finally retention days is one day. The important point here is that as the required reads and writes are 2, at least two nodes should be up and running.

2.2.3.3. server.properties

The information about each node is kept in a file named *server.properties* in *config* directory. This file contains some parameters to configure each node in Voldemort. In Figure (7), a *server.properties* file is shown:

```
node.id=0

max.threads=100

################ DB options ######################

http.enable=true
socket.enable=true

# BDB
bdb.write.transactions=false
bdb.flush.transactions=false
bdb.cache.size=1G

# Mysql
mysql.host=localhost
mysql.port=1521
mysql.user=root
mysql.password=3306
mysql.database=test

#NIO connector settings.
enable.nio.connector=false

storage.configs=voldemort.store.bdb.BdbStorageConfiguration, voldemort.store.readonly.ReadOnlyStorageConfiguration
```

Figure 7 : An example for server.properties file

As we can see, this is the *server.properties* file for the node with id=0. The max client threads that it can handle are 100. http data server and socket data server are enable in this node.

bdb.write.transacation= false means that, the write transactions are not immediately written on the disk. bdb.flush.transacations=false, means that when the write transaction is written to the disk, the disk is not forces to flush the OS cache which is an expensive operation. Moreover, the bdb.cache.size is 1 GB. We can use mysql or other storage engines as well, instead of bdb. host, port, user, password and the name of the database of mysql can be set here as well. enable.nio.connector=false means that the non-blocking socket connection is not supported. The classes for the storage configuration can be set here too.

2.2.4. Voldemort Rebalancing

Rebalancing is the one of the most important features of the Voldemort. There is a rebalance tool in Voldemort that adds/removes nodes from the cluster of nodes. There are some requirements during rebalancing that should be considered:

- No down time
- No functional impact on client
- Data consistency
- Minimal and tunable performance effect on the cluster
- Push button user interface

Furthermore, there are some assumptions for rebalancing:

- Only one rebalancing process should be running at a time

- The number of partitions is not changed during rebalancing, rather, partitions move to other nodes. In other words, partition ownership is changed during rebalancing.
- User should create a target cluster metadata.

2.2.4.1. Terminology

The process of rebalancing includes three participants:

- **Controller**: Starts the process by using the target cluster. The process can be started on any node in the cluster.
- **Stealer Nodes**: which are nodes that get partitions take moving partitions. It includes the nodes that are added to the cluster.
- **Donor Nodes**: Nodes that give partitions to other nodes. (Data is copied or moved from them to stealer nodes.)

2.2.4.2. Rebalancing Steps

In this section, we are going to show what is done in each steps of rebalancing:

- **Input** : Mandatory items for input are :
  - Target Cluster
  - The address of a node that rebalancing starts from.
  - The name of the store
- **Retrieve** the last state of the current cluster and compare the target cluster with.
- Add all new nodes to the current cluster. (Or removing nodes from the current cluster in case of removing nodes.)
- **Verify:**
  - Voldemort is not in the rebalancing state as only one instance of rebalancing is allowed at one time
  - Read only stores[7] use the latest version of storage format, if exist.
- For every “batch” of primary partitions to move
  - A transition cluster metadata is created.
  - A rebalancing plan based on this transition cluster.xml and the current state of the cluster is created. The generated plan is the map of stealer node to the list of donor node + partitions that should be moved.
  - Change the state on all stealer nodes.
- Start migrations of data units from read only stores. At the end of each immigration process, rebalancing state is deleted.
- Start migrations of data units from read write stores.

[7] There is a type of store in Voldemort that lets you build its index and data files in an offline system (like Hadoop) and once built, it provides a mechanism for swapping its current store for a fresher version.[12]

Rebalancing is done by running a shell file which is located in the bin directory named "voldemort-rebalance.sh". Table (4), shows the parameters for the Voldemort rebalancing:

Table 4 : Some important parameters used in rebalancing

Parameter	Description	Required/Optional
url	The name of the host or IP of one of the nodes to connect to	Required
target-cluster	The address of the target cluster file	Required
tries	maximum number of tries for every rebalance	Optional
parallelism	number of rebalances to run in parallel	Optional
no-delete	Do not delete data after rebalancing (Valid only for RW Stores)	Optional

2.2.5. Failure Detector

There are three types of failure detectors which are supported:

- **Bannage Period Failure Detector**: Triggers when the attempt to access the stores of nodes is failed and recordException is thrown. After this exception is thrown, the node is marked for a period of time as defined by client or server configuration. It is considered as available while this period passed. It means that it is available for attempting to access while it may be still down. If it is available to access the caller invokes recordSuccess and the node is marked as available.
- **AsyncRecovery Failure Detector:** It detects the node failure and then attempts to access the node's store to verify availability that may take several seconds. Therefore, this process is done in a thread in background instead of blocking the main thread.
- **Threshold Failure Detector:** It is the default failure detector of the Voldemort that has been built on the top of the AsyncRecoveryFailureDetector. Basically, it keeps the track of "success ratio" for each node that is the ratio of the successful operation to total operations. For mark a node as an available one, it is needed to its success ratio be equal to or more than a threshold. In every call of recordException, total operations are increased and successful operations remains constant. In each call of recordSuccess, total number of operations and successful operations is increased. What is more, the minimum number of requests should be done, until success ratio compared to the threshold. There is also a threshold interval in a way that the success ratio of a node is valid for a period of time.

2.2.6. Summary

The studied distributes storage systems are summarized in Table (5).

Table 5 : Summary of Distributed Storage Systems

DSS	Data model	Architecture	Consistency	Elasticity		
				Downtime	Automatic Load balance	Automatic Process
Cassandra	column oriented	cluster of nodes	Eventual	No	Semi	Yes
Hbase	column-oriented	cluster of nodes	Strong	Yes	No	Manual
MongoDB	document-oriented	collection of documents	Eventual	No	Yes	Yes
Voldemort	Key-Value	cluster of nodes	Eventual	No	No	Semi

We will use Voldemort as a distributed storage in our thesis. Voldemort is an open source key-value store which is inspired from Dynamo and Memcached [13]. Some motivations to choose Voldemort as our storage in our thesis are as following [15]:

- **No downtime during rebalancing**
- **No effect on the performance during rebalancing**
- **Maintain data consistency during rebalancing.**
- **Process of rebalancing:** As was mentioned, it is semi automatic. We should just configure the target cluster and the rest of the process is automatic.
- As we can see in Table (5), Voldemort is pure *key-value* storage. We intended to focus on a key-value store, Voldemort in the only case for us from this point of view.
- Good compatibility with YCSB
- Voldemort provides eventual consistency. It supports the lowest reads latency and Highest read throughput. Therefore, there is only the possibility of the reading old values. As we focus on latency (service time) of operations in our thesis and do not care too much to have the very last updated value, Voldemort is a good choice for us. Table (6), shows a comparison between eventual consistent read and consistent read.

Table 6 : Eventual Consistency vs. Strong Consistency (Adapted from [14])

Eventual Consistent Read	Consistent Read
Stale Reads Possible	No Stale Reads
Lowest Read Latency	Higher Read Latency

- Very clean code and Simple key/value API for the client

2.3. Yahoo Cloud Service Benchmark (YCSB)

As was mentioned before, we used YCSB [16] as a tool for benchmarking and evaluate the effect of adding/removing nodes on Voldemort while workload scenarios are exploited. There are two main concepts in this framework that we will discuss in following sections:

2.3.1. Benchmark Workload

In YCSB, there is a core set of workloads that is used to evaluate the performance of different systems that is called YCSB Core Package. Each package is a collection of related workloads. Each workload acts as multiple read/write operations with predefined data sizes and request distributions etc…

2.3.2. Distribution

When the workload client is going to generate workload, it should specify several options such as the type of operation to perform. These decisions are made by choosing random distributions:

- **Uniform**: choose an item uniformly at random. For example, when a record is going to be read, all records in the database are probable to be chosen equally.
- **Zipfian**: choose the records according to Zipfian distribution. When choosing a record, some of them are popular (the head of the distribution) and some them are not (the tail).
- **Latest**: Similar to Zipfian, but the most recent inserted records are at the head of distribution.
- **Multinomial**: Probabilities for each item can be predefined. For example, 0.95 for read operations and 0.05 for write operations.

Figure (8) shows the difference between uniform, Zipfian and latest distributions. The horizontal axes show the records that may be chosen and the vertical axes represent the probability of choosing an item.

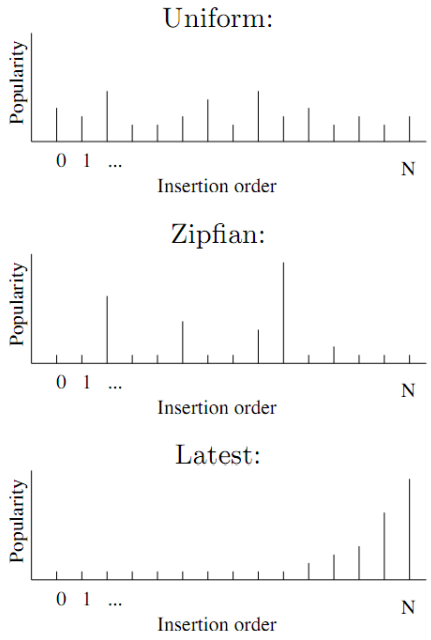

Figure 8 : Difference between distribution types in YCSB

2.3.3. Benchmark Tool

To execute benchmarks an open source tool that is called YCSB client has been developed. The most important characteristic of this tool is its extensibility which means that it can be used to benchmark all new Cloud database systems together with generating new types of workload. In the following section we will present the architecture of this tool in detail:

2.3.3.1. Architecture

The architecture of benchmark tool has been shown in Figure (9). The basic operation is the workload executor runs some client threads. Each thread operates a sequential series of operations by making calls to the database interface layer. They also measure the latency and the throughput that is achieved from operations.

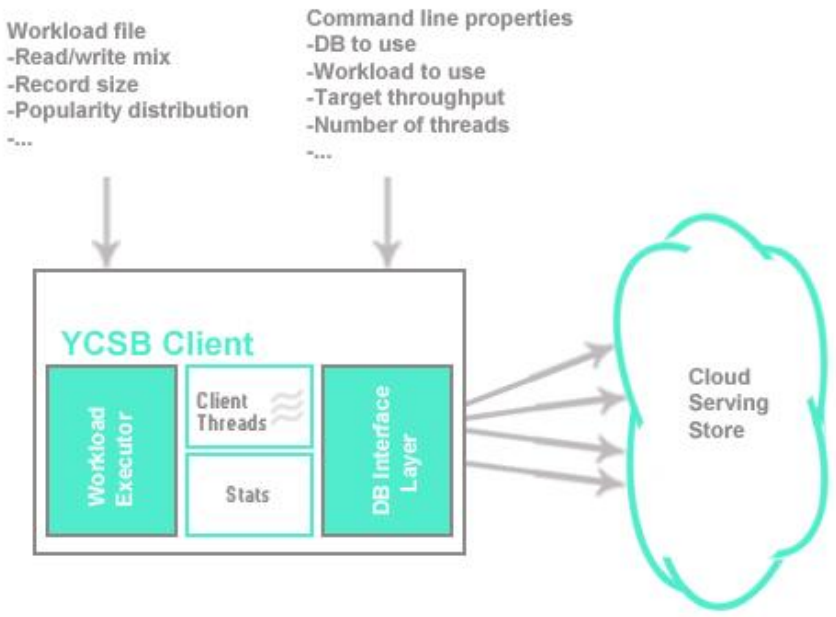

Figure 9 : YCSB client Architecture (Adopted from Figure (2) in [16])

To define the operations of client tool, it takes to main categories of properties:

- **Workload Properties**: Properties to specify how workload is going to be run without considering the type of database or experimental run. For example how to percentage of operations should be, the size and the number of fields in a record, total number of operations, target throughput, number of inserting records in the warm-up period, number of threads, the type of distribution.
- **Runtime Properties**: Specific properties of an experiment. Such as the hostname, the number of threads, the type of storage engine etc.

2.3.4. Using YCSB in Voldemort

For benchmarking, we use a shell file which is located in the bin directory of Voldemort project named “voldemort-performance-tool.sh”. Table (7), shows the parameters of benchmarking using YCSB [17]:

Table 7 : Some important parameters for benchmarking tool in Voldemort

Parameter	Description	Required/Optional
-url	The Voldemort server url (Example : tcp://hogwards.edu:6666)	Required

store-name	The name of the store	Required
ops-count	The total number of operations (delete, read, write, update)	Required
threads	This represents the number of client threads we use during the complete test	Optional
record-count	The total number of records to insert during the warm-up phase.	
iterations	While the warm-up phase can be run at most one time, the benchmark phase can be repeated multiple times. Default is 1	Optional
value-size	The size of the value in bytes. We use this during the warm-up phase to generate random values and also during write operations of the benchmark phase. Default is set to 1024 bytes.	Optional
d	execute delete operation [<percent> : 0 - 100]	Optional
r	execute read operation [<percent> : 0 - 100]	Optional
w	execute write operation [<percent> : 0 - 100]	Optional
m	execute update (read+update) operation [<percent> : 0 - 100]	Optional

2.4. Elastic Computing

In the Cloud computing area, Elastic computing is a web service that provides resizable compute capacity in the Cloud. It allows the system to scale up and down quickly while it computes the requirements changes. From the economic point of view it is very cost-effective, because the end user only pays for the resources that he actually uses. Amazon-EC2 is a good example in this field [18].

2.4.1. Elasticity

Data in distributed databases might grow very fast from the size and the number of users point of view. This makes the storage problem very essential. They should not only scalable but also elastic. There are differences between these two concepts. Scalability means, the expansion ability of the system which is expected to reach in future. This naïve approach has a very

important drawback that the ultimate size of the system should be specified in advance. There is a big contradiction between this concept and the most important property of Cloud computing "pay as you go". Because in this approach the end user should pay for the ultimate size of the system which may never be used. Another disadvantage is, as scaling up means adding more physical resources; it might be not easy to scale down by removing them.

To avoid these problems, a new approach has become favorite in recent years called elasticity. In this approach the final size of the system is not predefined. But system reacts to the changes are happened, dynamically. In the case of increasing the load, a new instance is added to meet SLO. Again, if the load decreases, the extra instance will remove from the system. We can see the difference between two approaches in Figure (10). As we can see, in the first case the capacity is in always constant and most of the time has not been used. Moreover, the maximum load exceeded from the expected capacity. But in the elasticity case, we can see that capacity is changed due to the load in the system dynamically and just a small part of the capacity is not used [19].

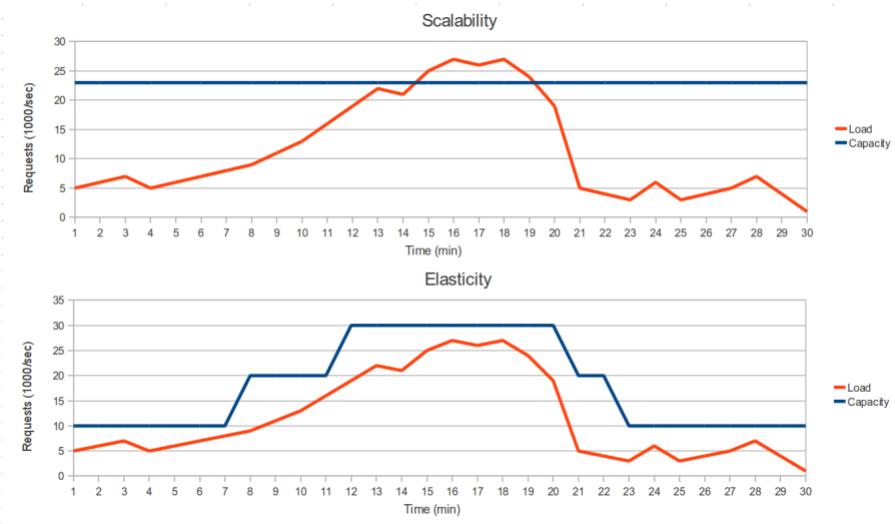

Figure 10 : Comparison of Scalability and Elasticity. Adopted from Figure I.1 of [19]

As you might have guessed already, the heart of such a dynamic system is a controller that monitors the changes in the system and reacts based on them to control the load to keep the system stable. We will discuss the concepts of control theory in section (2.7).

2.5. Service Level Agreement

Service Level Agreement [20] is "a formal contract used to guarantee that consumer's service quality expectation can be achieved". SLA was used since 1980's as customer satisfaction has been too important in utility computing. In other word, SLA defines the delivery ability of a provider, the performance target of consumers" requirement, the scope of guaranteed availability, and the measurement and reporting mechanism.

2.5.1. SLA Components

There are some components that have been defined for SLA. Some of them are discussed in the following:

- **Purpose**: Objectives to should be accomplished when SLA is used.
- **Restrictions**: Necessary steps or actions that need to be taken to ensure that the requested level of services are provided.
- **Validity period**: SLA working time period.
- **Scope**: Services that will be delivered to the consumers, and services that will not be covered in the SLA.
- **Parties**: Organizations and individuals that are involved in the SLA.
- **Service-level objectives (SLO):** Levels of services which both parties agree on. Some service level indicators such as availability, performance, and reliability are used.
- **Penalties**: If delivered service does not achieve SLOs or is below the performance measurement, some penalties will occur.
- **Optional services**: Services that are not mandatory but might be required.
- **Administration**: Processes that are used to guarantee the achievement of SLOs and the related organizational responsibilities for controlling these processes

2.6. Autonomic Computing

According to [19], autonomic computing can be defined as a method to develop self-managed complex software systems. It exploits some autonomic managers that implement some feedback control loops to monitor and control the behavior of the managed system. The properties of such computing are:

- **Self-Configuration**: automatic configuration of components. Figure (11), visualizes the idea of self-Configuration
- **Self-healing** : automatic discovery and correction of faults
- **Self-optimization** : automatic provisioning of resources
- **Self-protection** : automatic identification and protection from attack

Therefore, elasticity problem is somehow, self-optimization problem. Autonomic computing uses some methods to sense changes in a system and utilize them to reduce the cost or improve performance [21].

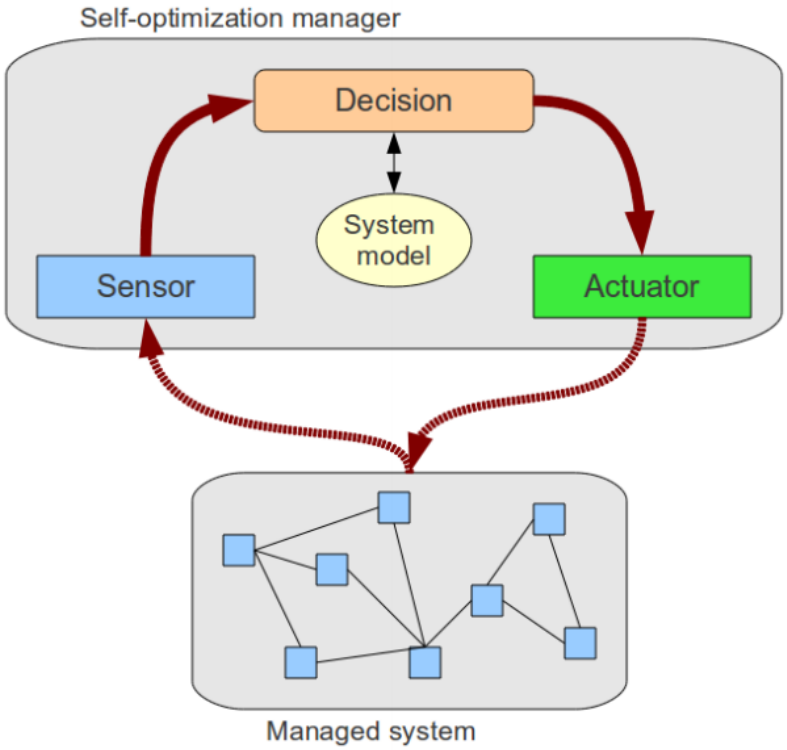

Figure 11 : Self-configuration control loop adopted from [19]

2.7. System identification and Control Theory

In this section we are going to discuss about two important concepts that we used in this thesis. In order to control a dynamic system we should have a model of the system from the behavior point of view. In the next sections system identification and control theory are explained respectively.

2.7.1. System identification

According to [22], system identification is “the art and science of building mathematical models of dynamic systems from observed input-output data”. It can be considered as a link between application in real world and model abstractions. It is a wide area that includes several techniques which using them depends on the properties of a system (model) that is supposed to be controlled or estimated such as linear, nonlinear, hybrid etc... . There are two major types of system identification:

- **First principle approach**: In this approach the properties of the system predicted by the physical laws or running the experiment. Although various first principles can be applied to determine a desired model, this approach has some shortcomings. The most

important drawback is the uncertainty that happens because of unknown parameters. Therefore this method is not used in complex systems with too many parameters as it is too difficult to find all parameters that effect on the system. This approach is better for gaining insight to the principles of the system.

- **Black-box approach:** In this approach, instead of going through the system and find out every detail of the system based on physical laws or knowledge, a mathematical approach is taken and a model is proposed based on input and output observation that includes enough description about the target system. This approach is widely used in designing controllers, because it reduces the efforts of the modeling. This method is also called empirical approach [23].

In this thesis we have used black box approach. Because our system is mostly a complex system and it was difficult to go through the system and specify all parameters that has effect on the output of the system. Therefore it was more reasonable to model the system without going through the system and focusing on the inputs and outputs.

2.7.2. Control Theory

According to [24],"Control theory is a mathematical description of how to act optimally to gain future reward". There are two major control systems:

- **Feed-Forward control systems**: This type of controller systems predicts relation between system and the environment variables and does some operations based on. The most important advantage of such systems is their speed. But they may cause instability in case of over-predicting.
- **Feedback control systems**: In these systems, information (outputs) from the process is used to improve the performance of the machine. In Figure (12), the block diagram of a feed-back control system has been shown.
  In the following, the components of these systems are discussed [25]:

  - **Reference input r (k)**: which SLO in our case that is specified by the user.
  - **Control error e(k)** : It is the difference between r(k) and output that is measured.
  - **Control input u(k)**: is the setting of one or more variables in the system that can be set by the controller.
  - **Controller**: calculates the values of the control input that is needed to be achieved by comparing current and past values of control error.
  - **Disturbance input d(k)**: is referred to any change that is caused influences the measured output.
  - **Measured output y(k)**: is the output of the system that is measured.
  - **noise input n(k):** is the noise that causes the any change emerged in the output of the system.
  - **Transducer**: Converts the measured output.

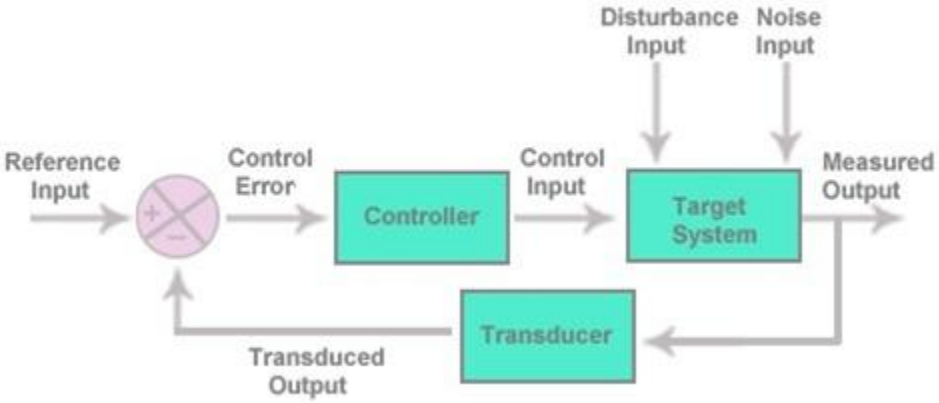

Figure 12 : A block diagram of a Feed-back Control System. Adopted from Figure 7.1 in [25]

2.7.3. Controller objectives

There have been several objectives defined for the controllers. In the following, some of important intended purposes of a controller are explained:

- **Regulatory control**: Guarantee that measured output of the system is equal or (mostly equal) to the r(k).
- **Disturbance rejection**: makes sure that disturbances cannot have significant changes on the measured output.
- **Optimization**: Acquire the best value for measured output of the system. For example obtaining the MaxClients in Apache HTTP server to minimize the response time.

Below we shall present the motivation of the properties was described:

- **Stable system**: A system is stable if for any limitation for input, the output is limited too. Stability is the most important property in the designing of control systems, because without it the unstable system cannot be used in critical works.
- **An Accurate control system:** is a system that the measured output value becomes close to reference input in case of disturbance rejection or regulatory control, or the measure output of the system become close to optimal value if the optimal value approach is used.
- **A system has "Short settling time":** that it quickly becomes close to its steady state that is too important for disturbance rejection when the workload changes.
- **A system should reach its goals without Overshoot:** that causes varying the measured output of the system.

These properties are called SASO in short.

2.7.4. Controller Types

In this section we will go through the several important types of the controllers that are being used. In this thesis, we used feedback controllers that are more popular in system controllers. Because as was mentioned earlier, in the feed-forward controllers the probability of instability is high due to over-predicting. Therefore, in the following we will present the different types of feedback controllers:

2.7.4.1. Proportional Controller (PC)

In this controller [26], the reaction which is done is directly proportional to the degree that system deviates from the ideal point. In other words, a gain can be considered as an amplifier to the controller, as it only serves to multiply the current error value by a given gain value which can be expressed as:

$$\mathbf{P_{out} = K_p e(t)} \qquad (1.1)$$

Which P_{out} is gain contribution parameter, Kp is Proportional gain, e(t) is the error term with respect to time. The equation of the e(t) can be presented as following :

$$\boldsymbol{e(t) = SP - PV}$$

*Which SP is the set point (*target value that system aims to reach*) and PV is the process variable.*

A large gain will cause a large change in the output of the system for a given error. Therefore, gain can be considered as an amplifier to increase the reaction speed of the controller to a certain condition. However, with high value of gain, system might become unstable quickly. On the other hand, if gain is too small, the controller response will be small to a given error and gives a controller with less sensitivity that may not react to the errors or disturbances.

2.7.4.2. Proportional-integral controller (PI Controller)

This controller has two parameters:

- **Proportional contribution** that specifies the reaction to the current error. As was mentioned above it equals to:

$$\mathbf{P_{out} = K_p e(t)} \qquad (1.2)$$

- **Integral contribution**: Calculates the system reaction based on the sum of recent errors. It can be displayed mathematically as following:

$$\mathbf{I_{out} = K_i \int_0^t e(t)dt} \qquad (1.3)$$

Which I_{out} is the Integral contribution parameter, K_i is Integral gain and e(t)is error term with respect to time.

2.7.4.3. Proportional-integral controller (PI Controller)

In this controller, in addition to $\mathbf{P_{out}}$ and $\mathbf{I_{out}}$, there is another parameter:

- **Derivative contribution**: Computes the degree at which the system error has been changing.

$$\mathbf{D_{out} = K_d \frac{d}{dt} e\ (t)} \quad (1.4)$$

D_{out} is Derivative contribution parameter, K_d is Derivative gain and e(t)is error term with respect to time.

In the other hand, PID algorithm can be written mathematically as following:

$$\mathbf{V_m(t) = P_{out} + I_{out} + D_{out}} \quad (1.5)$$

Therefore, from 1.2, 1.3 and 1.4, equation 1.5 can be rewritten as:

$$\mathbf{V_m(t) = K_p e(t) + K_i \int_0^t e(t)dt + = K_d \frac{d}{dt} e\ (t)}$$

2.8. Related works

There has been several projects and efforts done in the field of Elastic computing in Cloud computing. In this section we will look at two more interesting related works.

2.8.1. Automated Control for Elastic Control

In this paper [27], the goals are design and implement a controller for an elastic storage to monitor the load and scale up and down. They have used Hadoop Distributed Files System (HDFS) under dynamic web2.0 workloads. Moreover, Cloud Stone has been used for generating dynamic workloads. ORCA is used as the Cloud provider which is a resource control framework developed in Duke University.

In addition, the classical integral feedback controller has been used. This controller has 3 components:

- **Horizontal Scale Controller (HSC)**: that is responsible for growing and shrinking the number of nodes in the system.
  - Actuator : The number of active nodes in the system
  - Sensor: CPU utilization

The classical integral controller was used can be presented as an equation:

$$u_{K+1} = u_k + K_i \times (y_{ref} - y_k)$$

which u_k and u_{k+1} are the current and new actuator values (number of active nodes), y_k is the current sensor measurement. (CPU utilization), y_{ref} is the desired reference sensor measurement (CPU utilization), empirically, 20% average CPU utilization and K_i is the integral gain parameter.

- **Data Rebalance Controller (DRC):** that is responsible for controlling data transfers to rebalance the storage tier after it grows or shrinks.
  - **Actuator:** The bandwidth b allocated to the rebalancer. "The bandwidth allocation is the maximum amount of outgoing and incoming bandwidth that each storage node can devote to rebalancing."
  - **Sensor :** The cost of the system that is calculated by the following equation:

$$\text{Cost} = f_c(\text{Time, Impact}) = f_c(f_t(b, s), f_i(b, l))$$

    Which Time is the time of completion of rebalancing, b is the bandwidth throttle, s is the size of data to be moved and Impact is the effect of rebalancing on response time and l is the per-node workload

- **State machine**: To preserve stability during adjustments, the HSC and DRC must coordinate to manage their mutual dependencies. Figure (13), shows the state machine that of the elasticity controller for the storage layer.

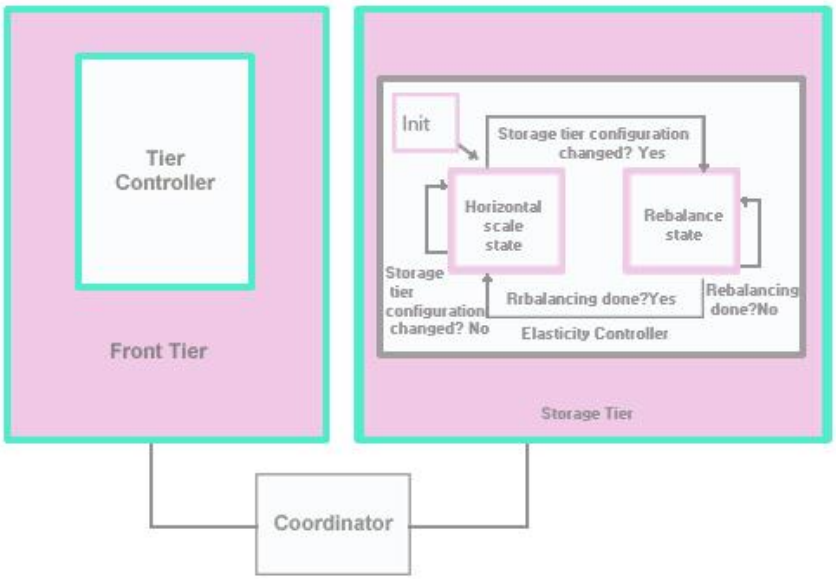

Figure 13 : The internal state machine of the elasticity controller of the storage tier. (Adopted from 5 in [27])

2.8.1.1. Implementation

Two experiments have been done in this work:

- **Adding node**: System starts with 6 nodes and after 10 minutes load gets ten times more and controller detects the change in the workload and adds another 3 nodes. Figure (14) shows Average CPU utilization of the HDFS data nodes with dynamic provisioning.

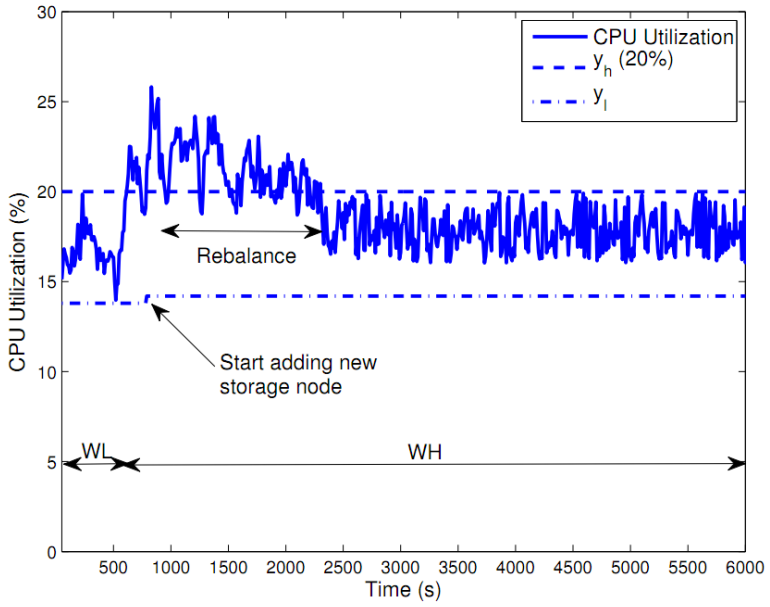

Figure 14 : The effect of dynamic provisioning on CPU utilization while workload increased. (Adopted from Figure 7c in [27])

- Removing nodes: System starts with some nodes. After 370 seconds workloads decreases 30%; After 50 seconds later, the controller senses this change in workload and removes some nodes. Figure (15), illustrates the scenario that was described.

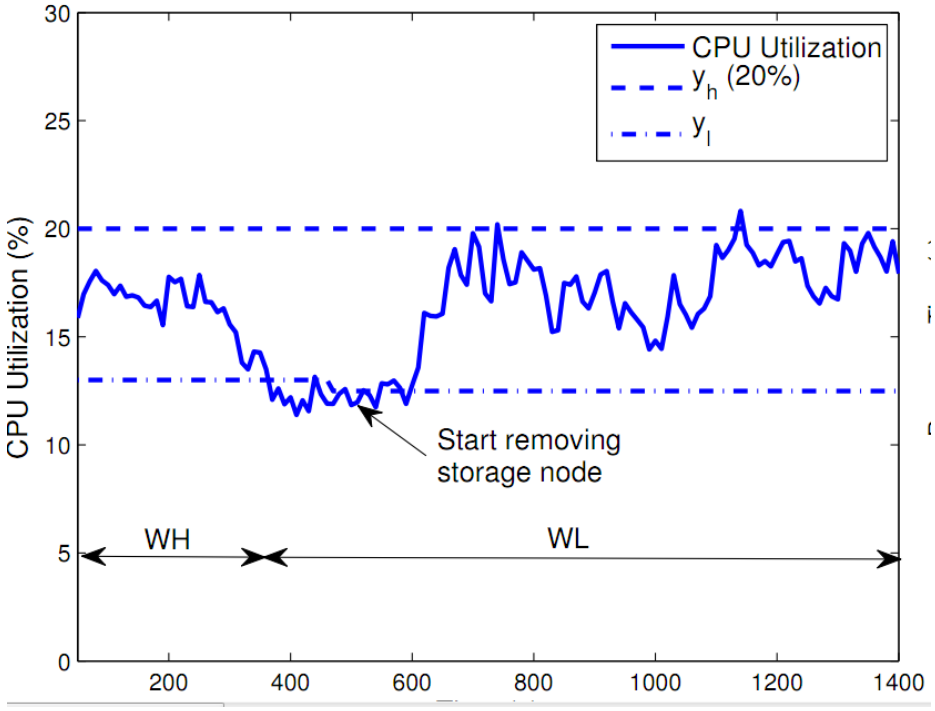

Figure 15 : The effect of dynamic provisioning on CPU utilization while workload decreased (Adopted from Figure 8c in [27])

2.8.2. Autonomic Management of Elastic Services in the Cloud

In this project [28], the goal is introducing a framework to manage the elastic services that includes a collection of agents that each of them follows the same pattern to handle events that come to them. An event is the occurrence of a situation, or incident, within a service or the management system, such as client arrival or failure of a transaction. We shall show this framework in 3 parts:

- **Service Management Model:** that is a set of constructs to specify management tasks.
- **Services Management Infrastructure**: Each managed resource that is the service or component is being managed exploits a WSDM [8] management endpoint. The management system can obtain performance metrics and adjust the configuration parameters. The metrics provided by each managed resource are obtained through the WSDM. Each agent is implemented as a WSDM entity which enables agents to communicate based on the standard protocols. All agents are combined to some goal graphs. In Figure (16), the architecture of an agent is illustrated:

[8] Web Services Distributed Management

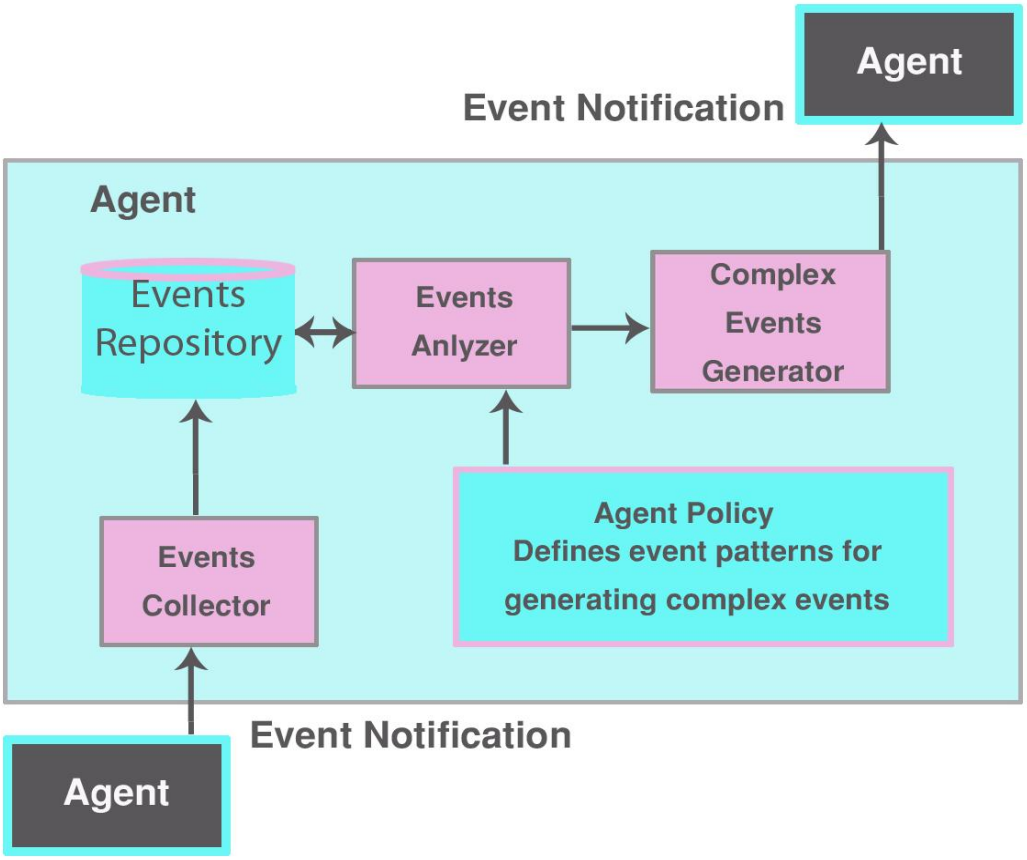

Figure 16 : Agent Architecture. (Adopted from Figure (1) in [28])

Moreover, there is another entity called effector that is created for each capability of the managed resource. It has the responsibility of the changes in the system by communicating with the endpoints and making a call to adjust one or more parameters in the system or to take some management actions on the managed resource.

- **Elastic Service Scenario**: A single virtual machine called delegator is introduced as the point of service access that acts as a load balancer that redirects incoming load across a pool of workers (VMs). The service applications are located in the VMs to accommodate the current load on the service. Elasticity here involves increasing and decreasing the number of worker instances. Figure (17) shows the elasticity scenario.

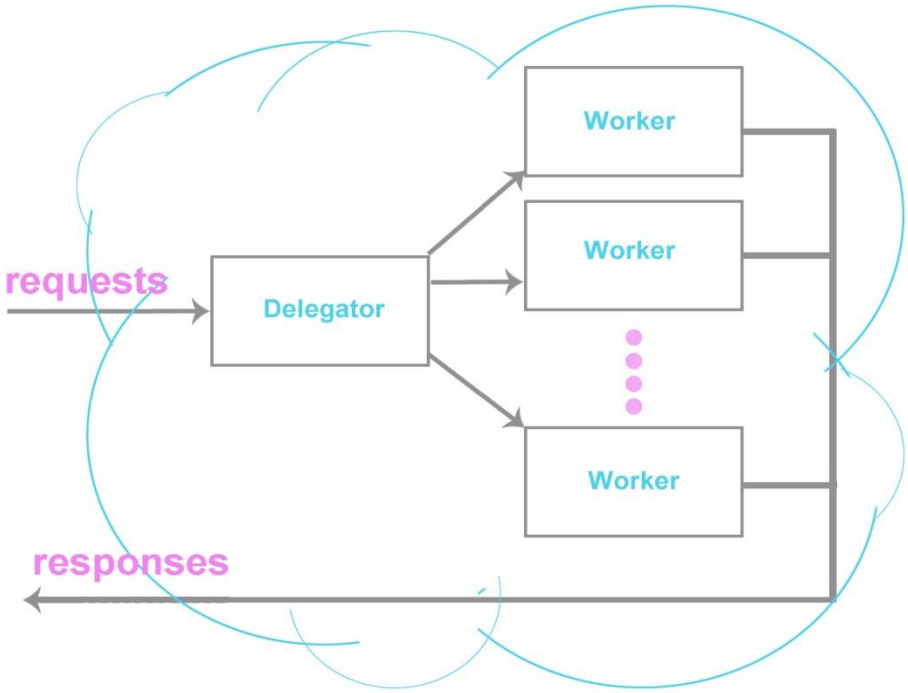

Figure 17 : Elasticity Scenario. (Adopted from Figure (2) in [28])

To support elasticity, the management framework is extended as we can see in Figure (18). In this figure *A* referents the Agent, *S* represents the Sensors and *E* represents the effector. Elastic services exploit the resources which are provided by IaaS layer of the Cloud which has been shown with the thick solid lines between managed sources (MRs) and Virtual Machines (VMs). Elasticity Goal Graph is another set of agents that act like Management Goal Graph and manage the elasticity as has been defined in the user-defined policy. Effectors in Elasticity Goal Graph carry out changes to the Management Goal Graph, the service and the basic resources allocated to the service.

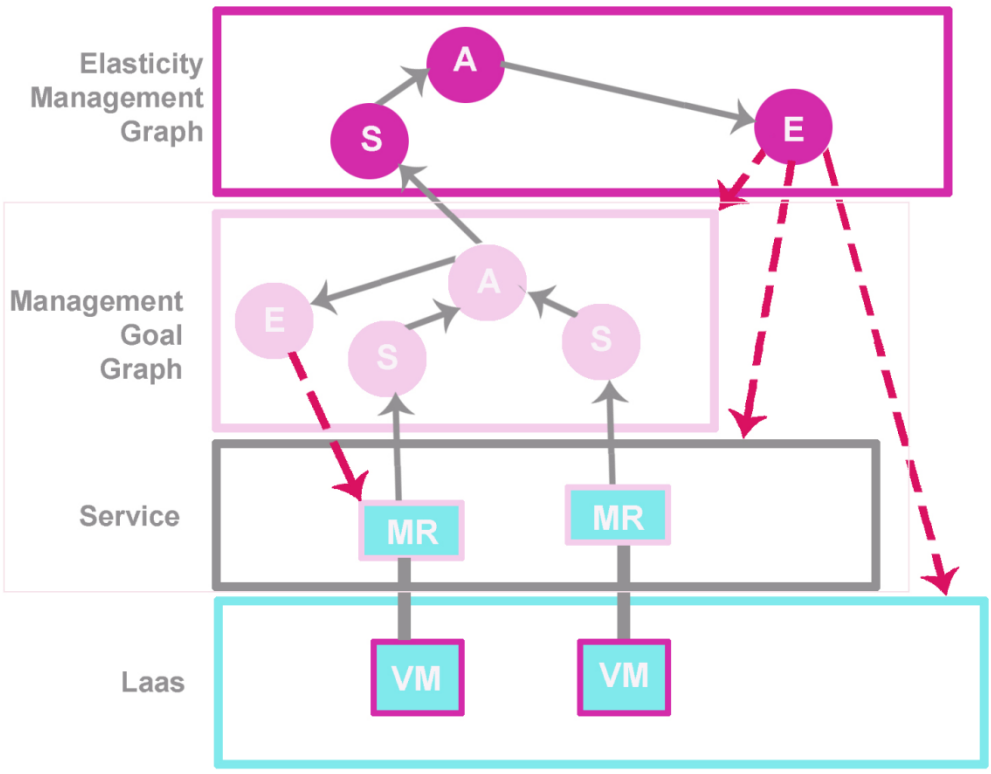

Figure 18 : Framework for Management of Elastic Services. (Adopted from Figure (3) in [28])

2.9. Summary

In this chapter we covered the concepts of the Cloud computing and discussed its advantages. We explained that Cloud computing enables the users to use elastic services. The idea behind Elastic computing is adding or removing resources for a service not to waste resources to reduce the cost of using them as well as high performance. In this thesis we focused on the storage services and used Voldemort as a distributed storage. To add or remove resources in an automatic way we should use Autonomic controller. To develop the controller, we mentioned the details about system identification and control theory. Moreover, to test and verify the system, we used YCSB that is a framework that provides us different load scenarios. Finally we discussed some important related works.

Chapter 3

3. Elastic Controlling Framework

In this chapter, we will explain the design of our controlling framework which could be considered as an extension for Voldemort storage in a way that it adds more resources (nodes) in case of increasing load and it removes some resources in case of decreasing load using an Autonomic controller. We shall first make a quick review of our problem with more details and then discuss about the design of our framework.

As we mentioned in the previous chapters, Cloud computing offers very flexible, high availability access to the resources, information, data, applications etc .. . Withal, it allows the users not to deal with the configuration of the hardware and software applications; rather they just plug to Cloud using the Internet and use the resources. Furthermore, Elastic computing is a controversial topic in IT area in recent years. All IT organizations tend to pay for the resources they use to be more conservative. In addition, more resources lead to more monitoring efforts and maintenance that increase the cost and is time consuming, too. Elastic computing provides resizable computing capacity in the Cloud that leads to using the resources while they are needed and paying for them as much as they are used.

We are going to design a controlling framework to implement this elasticity in a real distributed storage. Of course, that this control can be manual in a way that adding or releasing resources would be done by the administrator of the system. However, our framework has been designed to monitor the load in the system and grab or release some nodes based on the feedback controller that was designed and implemented. When the load increases, the nodes probably cannot handle the requests in appropriate time (SLO). Therefore, the controller would detect this and handle this issue by adding some more nodes according to its parameters. In other word, the role of the controller is deciding about the time of adding the nodes to the system and the number of nodes that are going to be added. Similarly, when the load decreases and the service time becomes less than SLO, the nodes are less busy and the storage can handle the requests with less number of nodes. Therefore by removing some nodes, we can save more resources and reduce the cost of using resources as a result.

In the following sections, the architecture of the controlling framework will be presented. We will cover design and implementation of the controlling framework. Finally we will discuss the system identification and different aspects of the controller design.

3.1. System Architecture

In the previous chapter, we understood that in a Cloud environment which is really dynamic, the management of resources becomes very important. They should be managed in some way that they would keep their efficiency. We are going to design and implement a controller that monitors the load in our storage system and requests for grab or release the nodes based on. In Figure (19), a generic control framework has been shown.

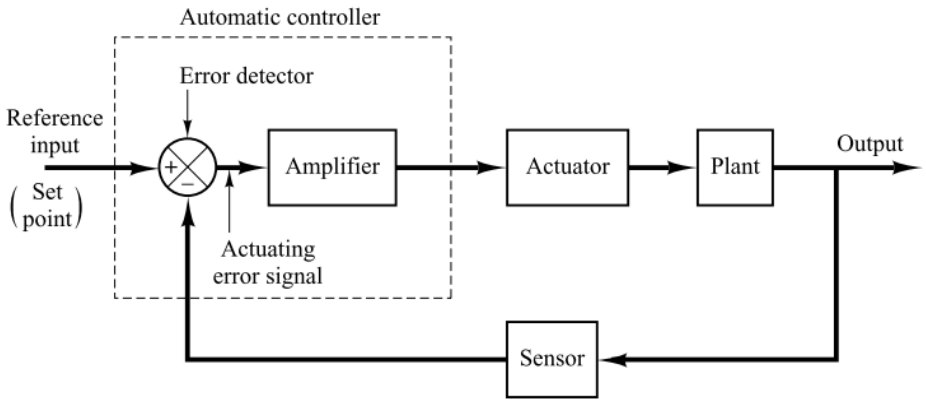

Figure 19 : Generic Control Framework (Adopted from Figure 2.6 in [29])

We can see that system has a reference input (Set point) which is the desired value of the user which is set by him. In our case, it is the desired service time that it is compared with measured output by the Sensor. This is done by the Error detector. The result is error signal which is amplified. This error signal is used by the Actuator which makes change in the plant (a typical system) that will result in the output change which is nearer to the Set point.

Now we consider our framework in more detail. We can consider the generic Architecture of the framework as shown in Figure (20):

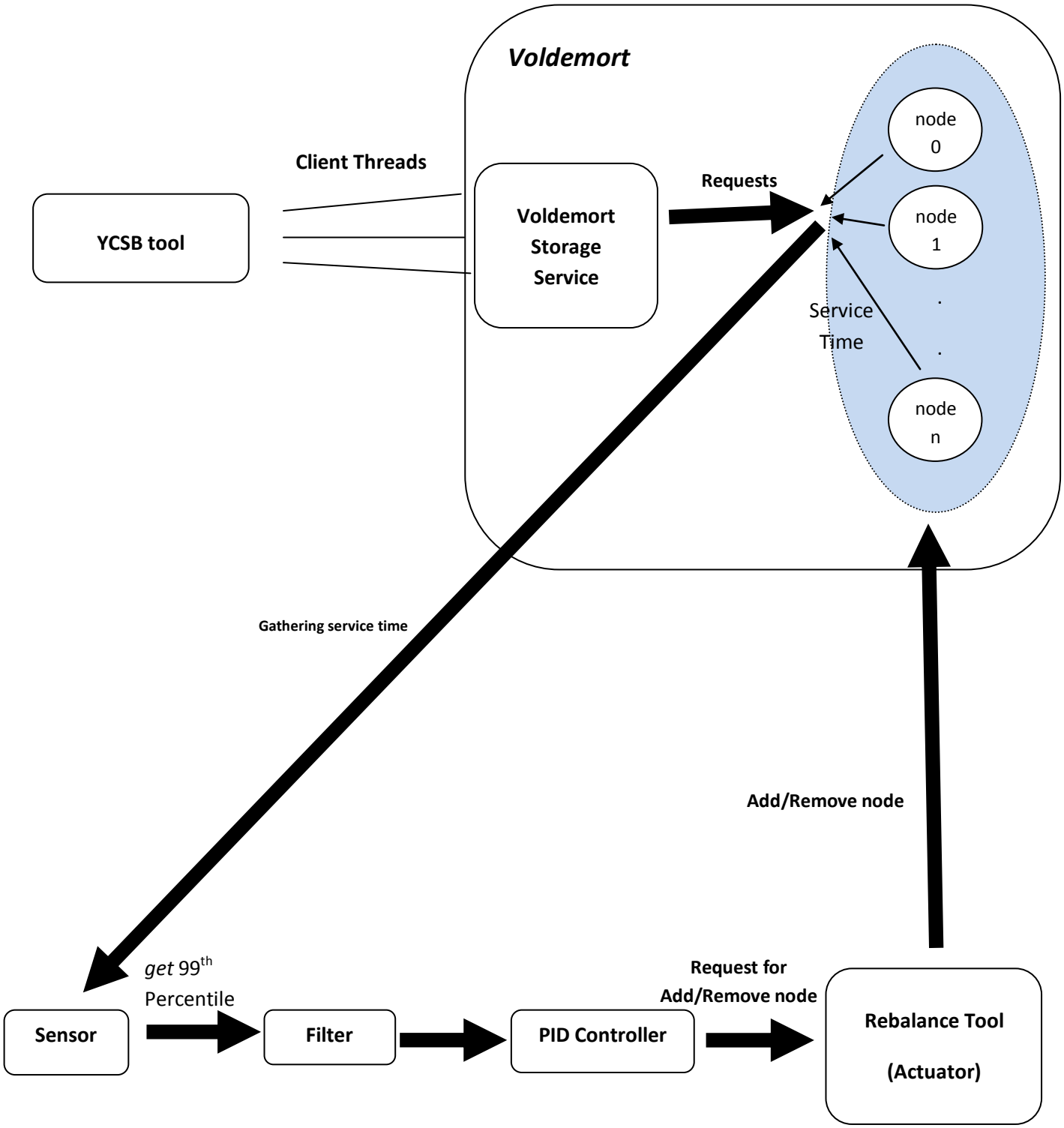

Figure 20 : Framework Architecture

As we can see there are six major components in our framework which are as following:

- **Voldemort**: A key/value storage that consists of a cluster of nodes. We discussed architecture, configuration and other aspects of the Voldemort in 2.2.1.4.
- **YCSB:** A benchmark tool which is an open source framework that creates various load scenarios. In 2.3, we covered design details and the usage of YCSB in our controlling framework.
- **Sensor**: Monitors the load by using average service time in each interval and gives it to the Filter. We will discuss more about the Sensor in (3.1.1).
- **Actuator (Rebalance tool)**: Gets a target cluster file from the PID controller and updates the cluster. We will present more details about the Actuator in (3.1.2).
- **Filter**: It causes the service time that is given to the controller smoother by avoiding spike output values from the Sensor. We will discuss more about it in (3.1.3).
- **PID controller**: Gets the average service time in each interval from the Filter and decides on the number of nodes that should be added or removed based on gain parameters that has been specified before. We will discuss about designing the controller in (3.1.4).

Table (8), shows how we designed and implemented or used each component of the controlling framework:

Table 8: Components in the controlling framework

Component	Implementation / tool
YCSB	Embedded benchmark tool in Voldemort
Actuator	Embedded Rebalance tool in Voldemort
Voldemort Storage	Java
PID Controller	Matlab/Java
Sensor	Java
Filter	Java

3.1.1. YCSB

As we mentioned in the previous chapter, YCSB is an open source framework which provides us some workload scenarios. There is a benchmark tool which has been embedded in the Voldemort for benchmarking is actually YCSB tool. In the following we are going to show that how we customized this tool to use in our thesis.

3.1.1.1. Implementation

There is a shell file in the *bin* directory of the Voldemort source called *voldemort-performance-tool* that runs the embedded rebalance tool in Voldemort. We ran this shell file on two powerful machined remotely via *ssh*.

3.1.2. Touch Points

According to [30], “is an interface to an instance of a managed resource such an operating system or a server. A touch point implements *sensor* and the effector behavior for the managed resource.”

In our thesis, we defined a touch point as an interface to Voldemort in a way that we could measure the service time in each node of Voldemort. Based on agreement, the service time of a request is the time that it is handled:

- ***put***
  - Reading the key and value from the input
  - Getting the version of the key that is going to be written.
  - Writing the key and value in storage which includes the time of writing on one or more nodes according to replication factor (in addition to routing time between nodes)
  - Returning the result
- ***get***
  - Reading the key from the input
  - Getting the Value and its version which includes the time of reading from one or more nodes based on the predefined “Required Reads”. (in addition to routing time between nodes)
  - Returning the result
- ***delete***
  - Reading the key from input
  - Deleting the value of the key which includes the time of deleting from one or more nodes based on replication degree. (in addition to routing time between nodes)
  - Returning the result

To measure the service time precisely, we used the math components of Apache Commons project [31] which basically is a library of Lightweight, self-contained mathematics and statistics components.

We put a math component on each node of Voldemort and took the service time of each operation (*put*, *get*, *delete*) and added it to *SynchronizedDescriptiveStatistics* object. At the end of each predefined interval (in our case 5 minutes), the controller is connected to each node and gets the statistics from them over socket. Using the collected *SynchronizedDescriptiveStatistics objects* from all nodes, we calculated several statistic results:

- put throughput
- get throughput
- delete throughput
- Total throughput

- Minimum put service time
- Minimum get service time
- Minimum delete service time
- Average Put service time
- Average get service time
- Average delete service time
- Maximum put service time
- Maximum get service time
- Maximum delete service time
- Total service time
- Variance of put service time
- Variance of get service time
- Variance of delete service time
- *put* 95th percentile time
- *get* 95th percentile time
- *delete* 95th percentile time
- *put* 99th percentile time
- *get* 99th percentile time
- *delete* 99th percentile time
- Total *put* operations
- Total *get* operations
- Total *delete* operations
- Total operations

We did several experiments and based on them we found out that average *get (read)* 99th percentile timein each interval gives us more reasonable and stable results.

3.1.2.1. Sensor Implementation

We used the touch points in order to monitor the load in the cluster of Voldemort nodes by measuring service time (which is average *get* 99th percentile time). In other word, the sensor in Figure (20) uses them to sense the load and give it to the Filter. In the following we are going to discuss about the classes which has been used to implement the sensor in detail. The next table shows the classes as well as packages:

Table 9 : Classes and packages used to implement Sensor

Class	Package
RunController	Controller
VoldemortServer	voldemort.server
VoldemortNativeRequestHandler	voldemort.server.protocol.vold
ServiceTimeSocketHandler	voldemort.server.protocol.vold
ServerSideServiceTimeStats	voldemort.server.protocol.vold
ServiceTimeThread	Controller
MonitorServiceTimeThread	Controller
Stats	Controller

ClusterInfo	voldemort.server

The major classes that involve implementing the sensor are:

- **RunController:** This is the main class for running the controlling framework. One of the important tasks of this class is creating some client threads as many as the number of total nodes in the cluster in each interval of sampling time. At the end of interval, each client thread connects to its server socket and gets a *SynchronizedDescriptiveStatistics* object which all parameters that mentioned in the previous section could be gained from.

- **VoldemortServer**: This class handles running the nodes in the Voldemort. We extended this class in a way that when a node runs, a server socket is created.
- **VoldemortNativeRequestHandler**: This class handles all types of requests (*put*, *get* and *delete*). We extended this class in a way that it also measures the time of each request and adds it to the *SynchronizedDescriptiveStatistics* object.
- **ServerSideServiceTimeStats:** This class act as an accumulator and the service time of every put, get and delete is added to SynchronizedDescriptiveStatistics, SynchronizedDescriptiveStatistics and SynchronizedDescriptiveStatistics respectively.
- **ServiceTimeSocketHandler**: In this class each server socket that has been created by VoldemortServer.java, listens to its related port. When a client socket connects to, it sends the *SynchronizedDescriptiveStatistics* over socket to the sensor.
- **MonitorServiceTimeThread:** This class gets each *SynchronizedDescriptiveStatistics object* has been sent from each node.
- **Stats**: This class gets the *SynchronizedDescriptiveStatistics object* extracts all parameters from and gives it to the controller. Here is where get 99th Percentile time is calculated in *getTotalAverageGetTime* function.
- **ClusterInfo:** provides information about the nodes such as hostname, port, nodeIDs, total number of nodes, etc...

3.1.2.2. Class Diagram

In the following Figure, we shall present a class diagram which shows the relation between classes which involve the implementation of the sensor. As was mentioned earlier, get 99th percentile time is calculated in *getTotalAverageGetTime()* function.

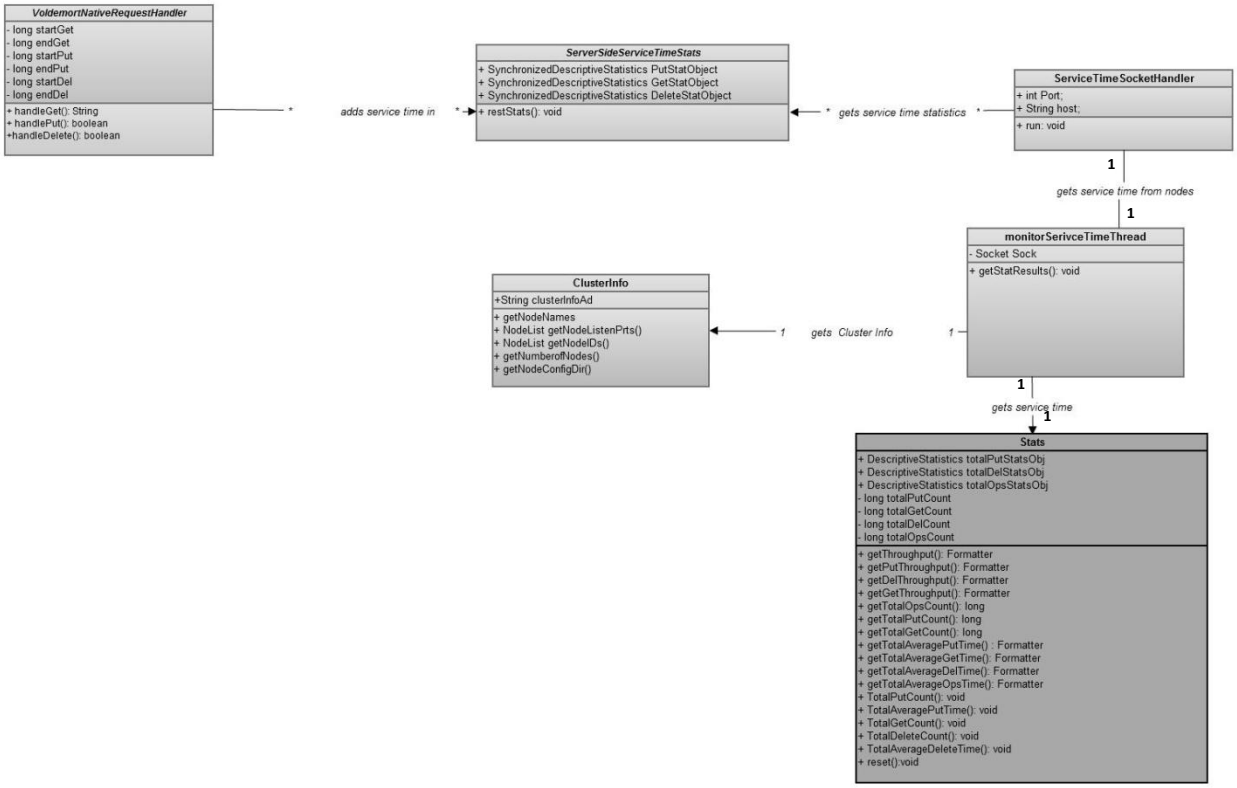

Figure 21: Sensor Class Diagram

3.1.3. Actuators

According to [32], an actuator can be seen as a system that takes the electrical energy and converts it to a mechanical motion. In the context of our thesis, an actuator is a process that can make some change in the storage that leads to system become a desired one. From our point of view, a desired system is the one that uses the resources based on the load changes. This is possible by adding or removing nodes in Voldemort. If the load increases, it will add some more resources to handle the increasing load and if the load decreases, it will removes some nodes not to waste the resources and save more money.

As was mentioned in the previous chapter, this adding and removing node is done during rebalancing process. Therefore we used rebalancing tool as an actuator.

3.1.3.1. Implementation

As we mentioned in the previous chapter, rebalancing in Voldemort requires configuring a target cluster before starting the process. For example if 3 nodes are going to be in the cluster, the target cluster is as following in Figure (22):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><cluster>
  <name>mycluster</name>
  <server>
    <id>0</id>
    <host>s1802.it.kth.se</host>
    <http-port>8081</http-port>
    <socket-port>6666</socket-port>
    <admin-port>6667</admin-port>
    <partitions>0, 1, 2, 3, 4</partitions>
  </server>
  <server>
    <id>1</id>
    <host>s1803.it.kth.se</host>
    <http-port>8092</http-port>
    <socket-port>6678</socket-port>
    <admin-port>6679</admin-port>
    <partitions>5,6,7,8,9,24,25,26,27,28,29,15,16,17,18,19,30</partitions>
  </server>
  <server>
    <id>2</id>
    <host>s1804.it.kth.se</host>
    <http-port>8104</http-port>
    <socket-port>6690</socket-port>
    <admin-port>6691</admin-port>
      <partitions>10,11,12,13,14,35,36,37,38,39,22,23,31,32,33,34,20,21</partitions>
  </server>

  <server>
    <id>3</id>
    <host>s1805.it.kth.se</host>
    <http-port>8116</http-port>
    <socket-port>6702</socket-port>
    <admin-port>6703</admin-port>
      <partitions/>
  </server>
```

Figure 22 : An example for a target cluster for rebalancing

We can see that there are 4 nodes in the cluster. No matter what is the current status of the cluster is, after rebalancing using this target cluster, the future cluster configuration will be like this. It means that the future cluster has 3 active nodes (nodes with partitions). It is possible to have a node without any partitions running in the cluster. But it does not contribute in the request handling processes.

As our nodes vary from 3 to 8, we configured 6 target clusters; each one is related to one state. Moreover, we configured 6 shell files that use these target clusters to run the rebalancing tool. Rebalancing process is handled in ***Rebalance*** class by calling function ***doRebalance()*** in a way that after the number of nodes that are going to be added or removed was specified by the controller, the rebalance tool is ran using related target cluster. For example, if total numbers of nodes that are going be in the cluster are 4, the target cluster with 4 active nodes (nodes with some partitions) is used as a parameter to rebalance the cluster.

3.1.3.2. Class Diagram

As we mentioned earlier, function *doRebalance(),* implements the Actuation process by getting the number of nodes that should be added or removed from the controller. If future total number of nodes in the cluster were more than maximum number of nodes (eight nodes), Actuator just would set total number of nodes to eight. Similarly, if future total numbers of nodes were less than minimum number of nodes (three nodes), actuator would set total number of nodes to three. Following figure shows the class that handles actuation.

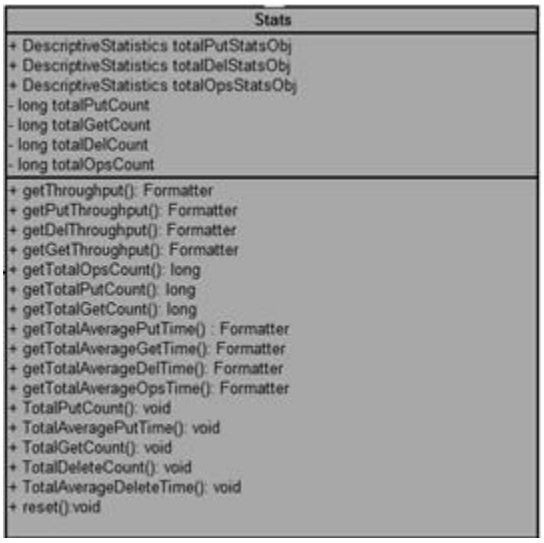

Figure 23: Actuator Class diagram

3.1.4. Filter

There are some times that measured values by the Sensor, are not smooth enough because of noise, failure or other special circumstances. We used a filter to reduce the influence of these undesirable situations. Therefore they can decrease the fluctuations in the measured output values.

We designed *Filter component* in a way that we gave more weight to the previous Filter output and less weight to the new filter input. We can show this concept in a mathematical way as following:

Filter output = 0.9*(Previous Filter Output) + 0.1 *(new Filter Input) (3.1)

We could see much more persistent values that were fed to the PID controller. Figure (24), shows a comparison between Sensor output values using a filter and without using it. Y-axis shows the *get* 99th percentile time in a sample experiment. We can see that after applying *Filter* to output values, we have smoother values.

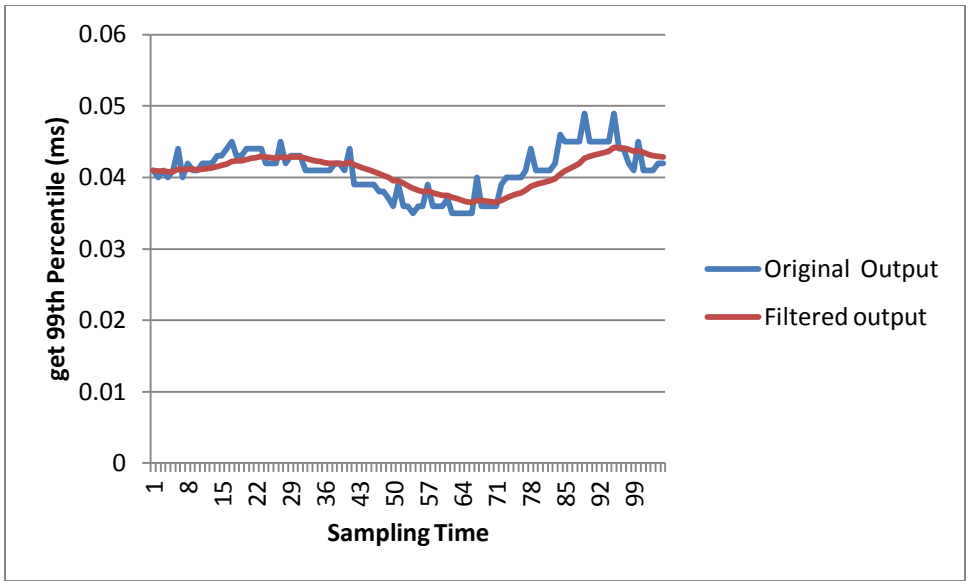

Figure 24: A comparison between output values using Filter and without using Filter

3.1.4.1. Implementation

There are two classes involve implantation of the Filter. In the following table their name and their package has been shown:

Table 10 : Classes and packages used to implement Filter

Class	Package
Stats	controller
Rabalance	controller

- **Stats**: This class gets the *SynchronizedDescriptiveStatistics object* extracts all parameters from and gives it to the controller.
- **Rebalance**: This class handles filtering process. This class gets 99th percentile time from Sensor and converts it to smoother value and gives it to the controller (in *getFilteredouput* function).

3.1.4.2. Class Diagram

In the following figure we shall present a class diagram that describes the classes that involve implementing the Filter and relation between them. As we mentioned earlier, function *getFilteredouput* implements filtering:

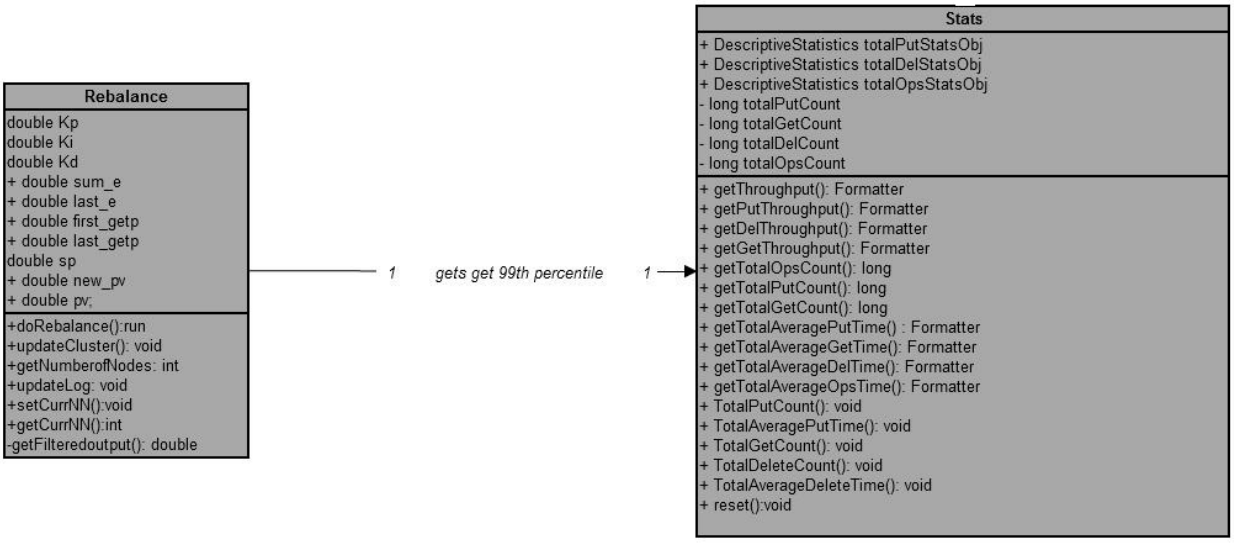

Figure 25 : Filter Class Diagram

3.1.5. PID Controller

One of the most important components in our controller framework is PID controller. It gets the filtered values form Filter component and decides how many nodes should be added or removed based on gains that have been calculated during controller design. In the following sections, first we will discuss about the system identification process and then the steps of designing the controller are presented in detail.

3.1.5.1. System Identification

Intuitively, system identification is the process of recognizing relation between the input and output of the system and how output depends on the input. More formally, it can be considered as a link between application in real world and model abstractions. As we mentioned in the previous chapter, there are two ways to system identification:

- **First principle approach**: that the properties of the system should be known almost completely and the system is predicted by physical laws or running experiments. This approach is not efficient for complex systems with too many properties.
- **Black-box approach:** which instead of knowing all properties of the system, a mathematical approach is taken and a model is proposed based on different input and output. We used this approach in our thesis as our system is mostly a complex system with unknown parameters.

In any system identification with black-box approach, there are some steps that should be followed:

- Determining the input/output of the system.

- Experiment and collect the input and output of the system which will be discussed in the next chapter.
- preprocess data and select the useful part of data
- Design a model based on the data which has been collected
- Observe the system behavior. If the model does not reflect the system behavior, go to the first step.

3.1.5.1.1. System input/output

Input and output of the system are shown in the Figure (26). As we can see in this Figure:

- System Input is the number of nodes that are going to be added or removed.
- System output is *get* 99th percentile time. As was mentioned in this chapter, we measured several output for the system using the apache commons components. After running various experiments, we found out that the *get* 99th[9] percentile time is the best parameter to represent the output of the system. Because it replied more reasonable results to the different load scenarios that we applied to Voldemort.

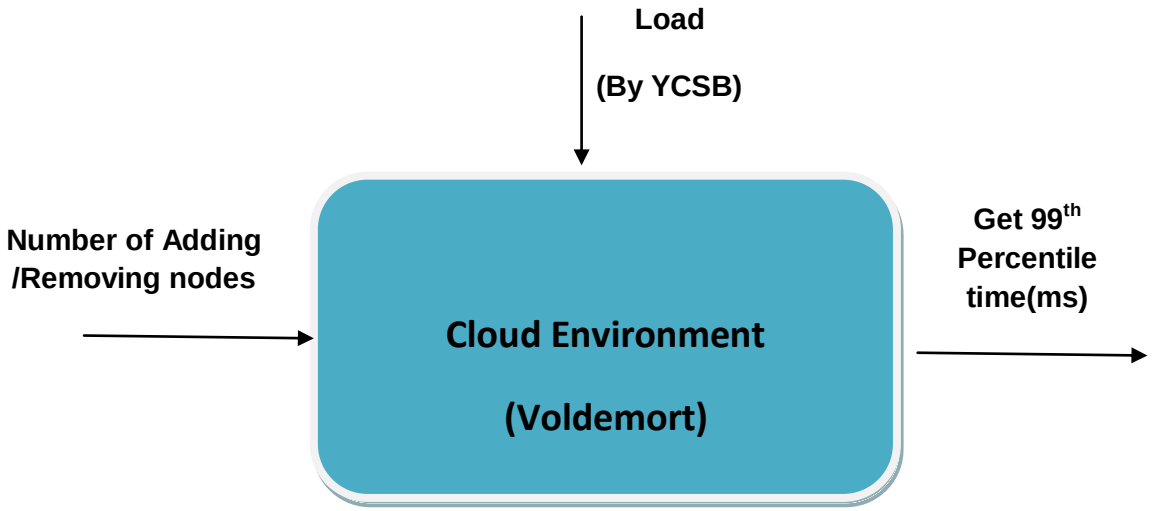

Figure 26 : System Input/output

3.1.5.1.2. State Space Model

We used System Identification Tool in Matlab to model the system. We import input and output data that we gained in the data acquisition step in this tool and choose linear parametric models

[9] Get 99th percentile = x, means that 99% of read operations are take below x (ms).

to estimate the model. Command *ident* opens this tool for us. There are two main approaches in the system modeling:

- **ARX**: which is the simplest model structure which is a linear difference equation that relates input and output of the system as following:

$$y(t) + a_1y(t-1) + ... + ana\ y(t-na) = b_1u(t-n_k) + ... + bnbu(t-n_k-nb+1) \quad (3.1)$$

There are 3 parameters which should be specified. na is the number of poles, nb+1 is the number of zeros and nk is the pure delay time. [33]:

- **State space:** which model a system based on the input, output and state variables within the system. The most important advantage of this approach is its extensibility in a way that we can add input and output to the system easily [33]. We chose this approach to model our system.

The dynamics of basic state space model is as following:

$$x(k+1)= Ax(k) + Bu(k) \quad (3.2)$$

$$y(k) = Cx(k) + Du(k) \quad (3.3)$$

x(k) is the vector of state variables, u(k)= output Matrix, Y(k) the Input Matrix , D(k) is the Delay Scalar.

System Identification Tool offers two approaches to model a system using state space.

- **PEM**: This is suitable for linear low order state space models which we used in our system modeling.
- **N4SID**: This approach is faster than PEM, but less accurate and robust. Moreover, it requires additional parameters which might be difficult to specify.

To use **PEM** approach, we should specify two parameters, delay and the order of the system. Delay is a vector of the number of input delays which is zero as is considered in discrete models. The order of the system is estimated by following command in Matlab. This command estimates the parameters for the state-space model:

```
pem(dat,'best')
```

This command specifies the best order for the system. ***dat*** is an object which is created by the idata command which takes the output vector and the input vector as its input. By using pem command, we found out that the best order for the system is 2.

Now we have all parameters to model the system. After adding the model object to the work space, now it is time to estimate initial steps of the model. It is specified by the following command:

pem(dat,'best','InitialState','estimate')

and we have the initial states as a scalar:

$$x_0 = \begin{bmatrix} 0.32689 & -0.96019 \end{bmatrix}$$

After we designed the model with system identification tool, we can specify A, B, C and D which specified in 3.2 and 3.3 equations with the state **ss** in Matlab. It creates some state-space object from PEM model that we have built.

sys=ss(pss)

pss is the PEM model object that was created by System Identification Tool.

$$\mathbf{A=} \begin{bmatrix} 0.88577 & -0.035944 \\ 0.11396 & 1.0356 \end{bmatrix}$$

$$\mathbf{B=}\begin{bmatrix} -0.00069035 & 0.00059463 \end{bmatrix}$$

$$\mathbf{C=}\begin{bmatrix} 0.142 & 0.0054066 \end{bmatrix}$$

$$\mathbf{D=}\begin{bmatrix} -9.1668e-005 \end{bmatrix}$$

3.1.5.1.3. Transfer Function

The next step in system identification process is building transfer function of the system which models a linear system by a transfer function of the Laplace-domain variable. It is calculated by the tf command in Matlab which takes the state space model object as input (which is taken from ss command). It converts state space model to transfer function form:

Transfer Function=tf(sys)

Therefore we have the transfer function of the system as following:

$$\frac{0.0001565\ z\text{^}2 - 9.465e{-}005\ z + 7.484e{-}006}{z\text{^}2 - 1.921\ z + 0.9215}$$

3.1.5.2. Controller design

We used *Simulink* as an approach to design the controller. Other approach could be using SISO (Single Input Single Output) tool. *Simulink* is a graphical environment which lets us to design, implement and verify the controller. The graphical designed controller in Simulink is as following:

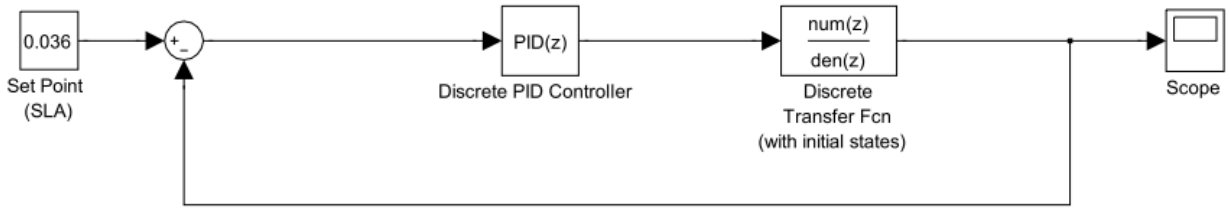

Figure 27: Graphical design of the PID controller using Simulink

To configure the transfer function block, we insert the dominator and numerator coefficients of the transfer function that was calculated to the block as well as initial state scalar. We set SLO (reference point) as 0.036. Then we can set the gain parameters K_p, K_i and K_d of the PID controller by tuning the controller. After tuning the gain parameters using PID Tuner, finally we reach the block response and Tuned response time (using PID controller):

Now we have the gain parameters of the PID controller:

$$\mathbf{K_p}: 1.19785394231464$$

$$\mathbf{K_i}: 0.0256625849637579$$

$$\mathbf{K_d}: -288.920114195685$$

3.1.5.3. Controller Implementation

Rabalance: This class handles implementing the PID controller. It gets the filtered values from Filter component and calculates the number of nodes to add or remove based on gain parameters that has been defined. As the output of the controller is a *double* value, we used ***Math.round()*** method to amplify the output. For example if the output of the controller was 1.837854758 the amplified output would be 2. Because we probably would get better results by adding 2 nodes instead of one. Similarly, if the output of controller was -1.294857487, the amplified output would be -1. Because the results would be better by removing one node instead of two.

3.1.5.4. Class diagram

The following class diagram shows the relation between classes that involve the implementation of the PID controller. Function *getNumberofNodes()* handles the implementing the controller.

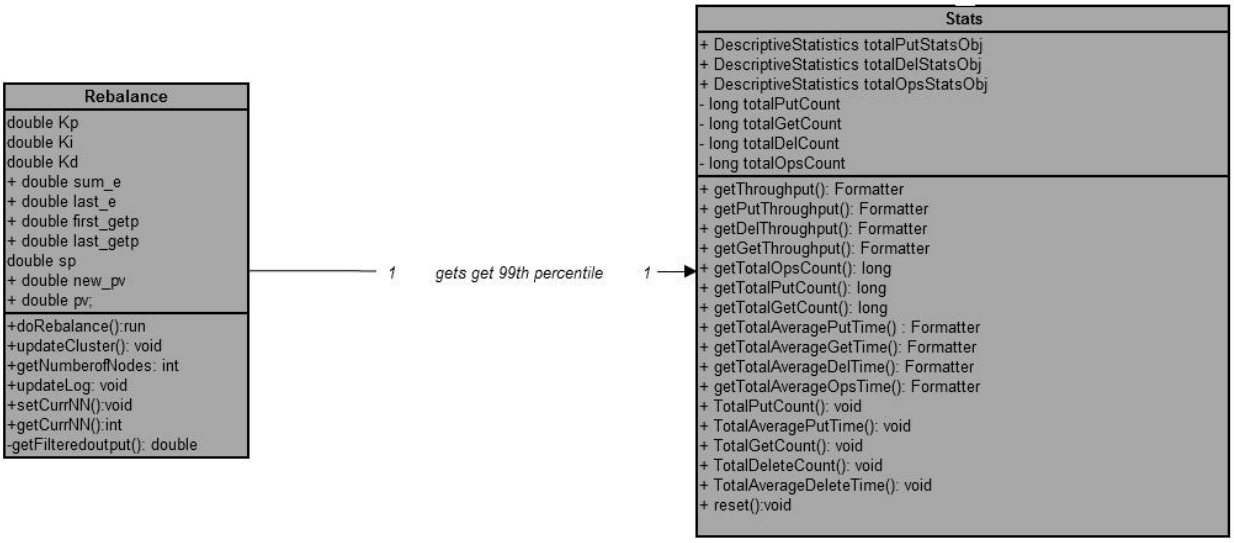

Figure 28 : Controller Class Diagram

3.2. Summary

In this chapter we covered the design and implementation of controlling framework. We discussed about the controlling framework components the role of them in the framework as well as how we implemented them with java. Moreover, we explained the process of the system identification, system modeling and designing the controller using *Simulink. We presented how we modeled the system using System Identification Tool in Matlab and then calculated the parameters of the controller.*

Chapter 4

4. Evaluation and Experimental Results

In the previous chapters, we described the problem that we focused on. We covered the background of the problem, related works, the architecture of our controlling framework and some details about the system identification, system modeling and PID controller design.

In this chapter, we shall continue with identification of the system, and perform experiments and evaluation the result.

4.1. Experimental design and data acquisition

As we saw in the previous chapter, the input of the system is the number of the nodes that are going to be added or removed in the system and the output is the average *get* 99^{th} percentile time. We also mentioned that the second step in the system identification is data acquisition. In the next section we shall show how we gathered data to identify the system.

4.2. Setup

We discuss the experimental setup in two parts, node setup and benchmark setup. In both sections, the parameters selected empirically by running various experiments led us to the most efficient parameters:

4.2.1. node Setup

Our cluster consisted from eight nodes running on eight machines. In the following table, the node setup of our experiment is presented:

Table 11 : node Setup for data acquisition

Machine Setup	
Parameter	**Value**
Processor	4
CPU Cores	4
model name	Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz
Cache size (MB)	3
Memory (MB)	3887
Swap (MB)	8001
Node Setup	
Parameter	**Value**
Voldemort Version	0.90.1
Database Server	Berkely DB
Socket Timeout (ms)	90000
Routing Timeout (ms)	100000
Bdb cache size	1 G
JVM_SIZE (Min and Max)	4096 MB
Replication Factor	3
Required Writes	2
Required Reads	2
Key Serializer's Type	String
Value Serializer's Type	String

4.2.2. Benchmark Setup

For effective benchmarking, we used two powerful machines to load the Voldemort nodes as much as benchmark parameters were set. Table (12), shows the setup used in our benchmark:

Table 12 : Benchmark Setup

Machine Setup	
Parameter	**Value**
Processor	24
CPU Cores	6
Model Name	Intel(R) Xeon(R) CPU , X5660 @ 2.80GHz
Cache Size (MB)	12
Memory (MB)	44255
Swap	20002
Benchmark Setup	
Parameter	**Value**
Number of records inserted in the warm-up period	10000

Write Percentage (%)	5
Read (%)	95
Showing Result Interval (Sec)	60
Throughput (Ops/Sec)	4000
Sampling Time (min)	5

4.2.3. Benchmark Experiment

We started with three active nodes and run the controller and two YCSB instances with a specific throughput. Then with the delay D=30 (min) between each rebalancing, node 4th, 5th, 6th, 7th and 8th were added. Afterwards, with the same delay, node 8th, 7th, 6th, 5th, 4th are removed to cover the all range of input.

Figures (29) and (30) show the results of experimental design and data acquisition. The X-axis shows the sampling time. Figure (29), Y-axis shows the number of nodes and In Figure (30) shows the average *get* 99th percentile time. As we can see, by increasing the number of nodes, the *get* 99th percentile timedecreases. Similarly, by removing some nodes, *get* 99th percentile time increases.

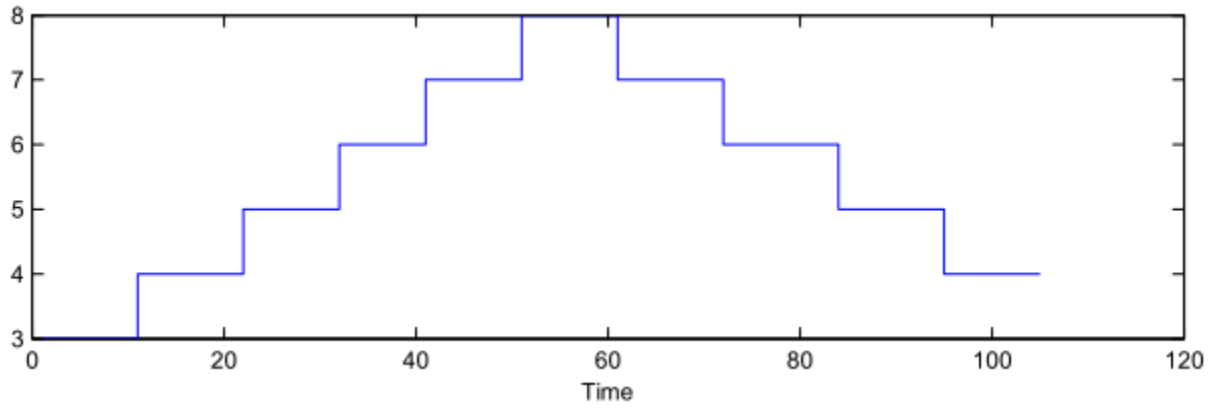

Figure 29 : The changes in the number of nodes in the experimental design and data acquisition

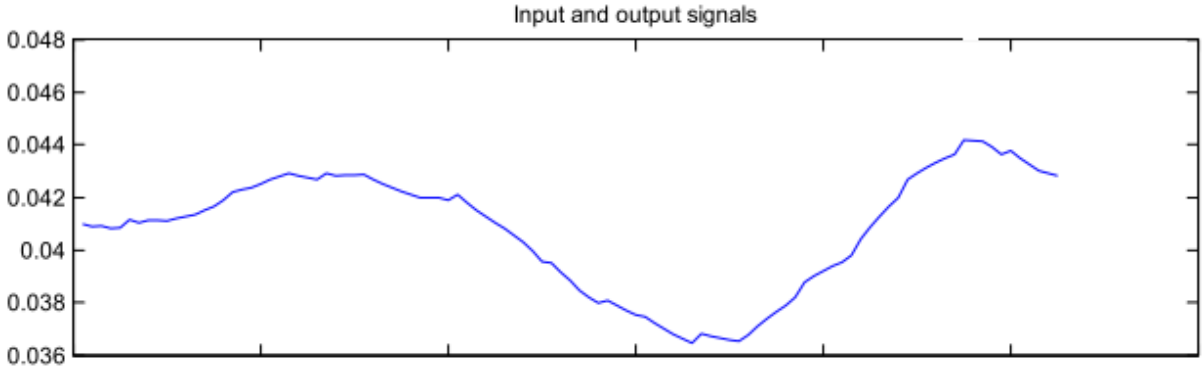

Figure 30 : The changes in get 99th percentile timechanges in the experimental design and data acquisition

As was mentioned in the previous chapter, the model that is created by system identification tool is a *PEM* model in State Space structure. Figure (31), shows the output model which compares the measured outputs and simulated model. Y-Axis shows the *get* 99th percentile time. As we can see that there is a good consistency between measured and simulated model.

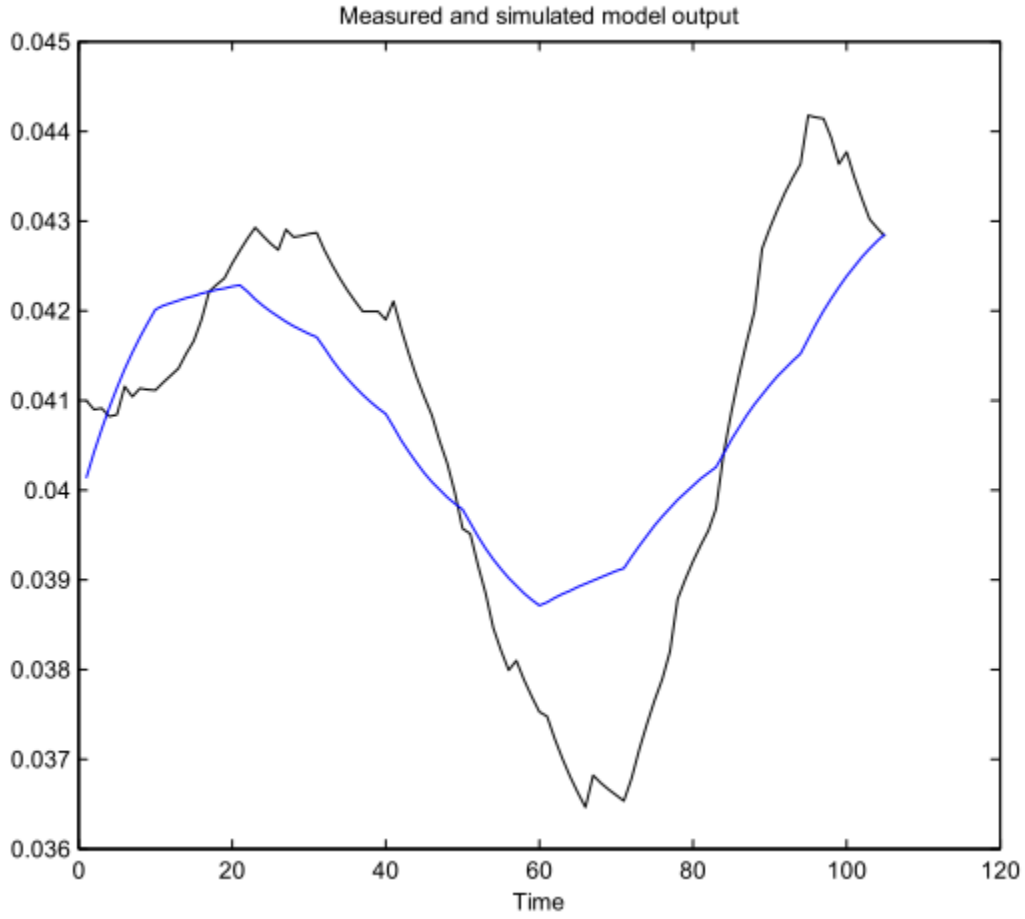

Figure 31 : Model output. The black curve shows the measured output values and the blue one shows the output of simulated model by Matlab

4.3. Experiment using the controller and Evaluation

In this section, we shall present the experiments that we have done for testing our designed controller on Voldemort using YCSB as workload generator. In the following, we first shall show how we verify the precision of the controller by simulating the average *get* 99th percentile time values and then we go through the real experiments on Voldemort.

4.3.1. Controller verification

We did some simulation experiments to see the accuracy of the designed controller. In other word, we fed the PID controller some values as the average *get* 99th percentile time and then ran the controller which was implemented in Java with gain parameters that was defined in the previous chapter. Table (13), reviews the controller properties.

Table 13: Controller Parameters

Parameter	Value
Type	PID
Kp	1.19785394231464;
Ki	0.0256625849637579;
Kd	-288.920114195685;
SLO (ms)	0.036

Figures (32) and (33) show the results of this experiment. As we can see, the controller responds to the changes in the average *get* 99th percentile timewell enough. In other word, it adds enough nodes to handle the load in case of increasing average *get* 99th percentile time. Moreover, it removes reasonable nodes in case of decreasing average *get* 99th percentile time. This shows that the designed controller is precise enough to handle the load in Voldemort.

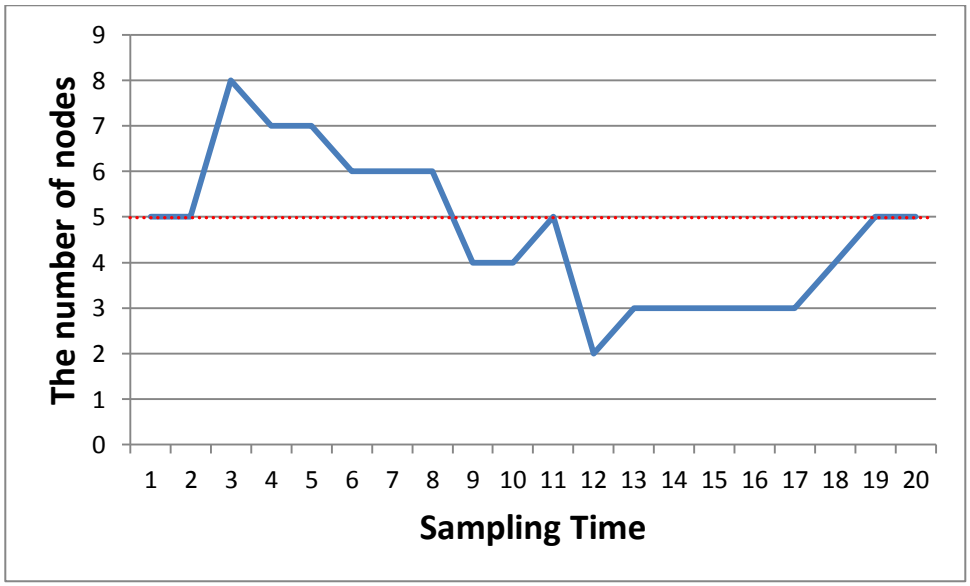

Figure 32 : The changes in the number of nodes in the controller verification

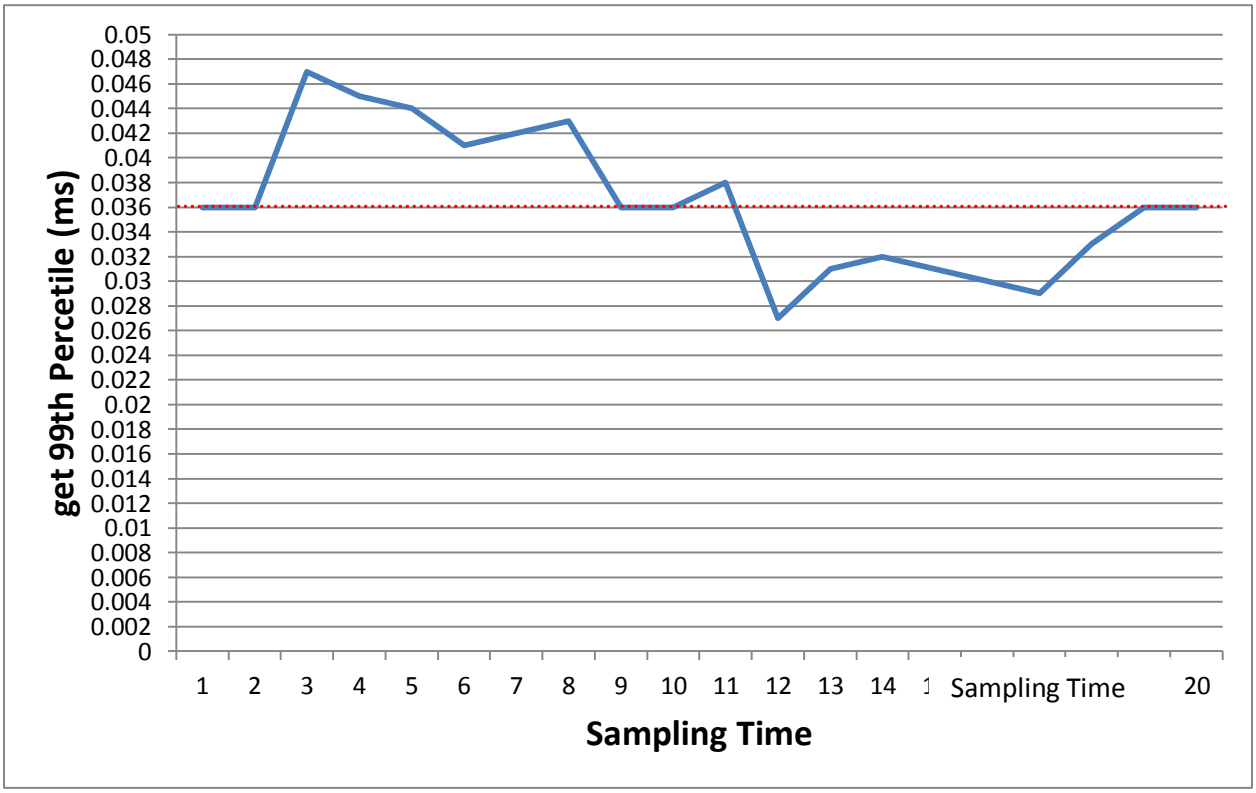

Figure 33 : The changes in the average get 99t^h percentile time during controller verification

4.3.2. Experiment using YCSB on Voldemort

In this section, we shall present how we ran experiments to test the controller in a real environment. We ran three experiments to show how the controller responds to the load changes. In the first experiment we showed how controller works in case of higher *get* 99th percentile timethan SLO which is 0.036 (ms) in our case and in the second experiment we showed that how controller reflects against the values less than SLO. Finally in the third experiment, we shall show that how the controller works in the case of changing the load.

4.3.2.1. Experiment Setup

Table (14), shows the experiment setup in detail. We ran the sensor for one hour as a warm-up period. After warm-up period the PID controller was run.

Table 14 : node setup for experiments

Machine Setup	
Parameter	**Value**
Processor	4

CPU Cores	4
model name	Intel(R) Core(TM)2 Quad CPU Q9400 @ 2.66GHz
Cache size (MB)	3
Memory (MB)	3887
Swap (MB)	8001

Node Setup	
Parameter	**Value**
Total number of nodes	8
Initial number of nodes	5
Minimum number of nodes	3
Voldemort Version	0.90.1
Database Server	Berkely DB
Socket Timeout (ms)	90000
Routing Timeout (ms)	100000
Bdb cache size	1 G
JVM_SIZE (Min and Max)	4096 MB
Replication Factor	3
Required Writes	2
Required Reads	2
Key Serializer's Type	String
Value Serializer's Type	String

We used two machines to run benchmark workloads. Table (15), shows the configuration of each machine:

Table 15 : Benchmark setup for experiments

Machine Setup	
Parameter	**Value**
Processor	24
CPU Cores	6
Model Name	Intel(R) Xeon(R) CPU , X5660 @ 2.80GHz
Cache Size (MB)	12
Memory (MB)	44255
Swap	20002

Benchmark Setup	
Parameter	**Value**
Number of records inserted in the warm-up period	10000
Write Percentage (%)	5
Read (%)	95
Showing Result Interval (Sec)	60
Throughput (Ops/Sec)	4000
Sampling Time (min)	5
Used keys	1-1000

In our experiments, for simplicity, sampling time is fixed and equal to 5 minutes. However, we could consider some flexible sampling strategy that either of them could be applied due to end users' behavior. For example, during day which the load is high, we could apply sampling strategy with shorter time to handle the load faster.

4.3.2.2. First Experiment

Figures (34) and (35) show the result of first experiment. We can see that after warm-up period the controller starts at point ***C***, the average service time is more than SLO (0.036 (ms)). Therefore, after controller started, it adds enough nodes to handle the workload. The controller adds three nodes, so total nodes=8 which is the maximum number of nodes. As a result, the number the average *get* 99th percentile timetime becomes closer to the SLO.

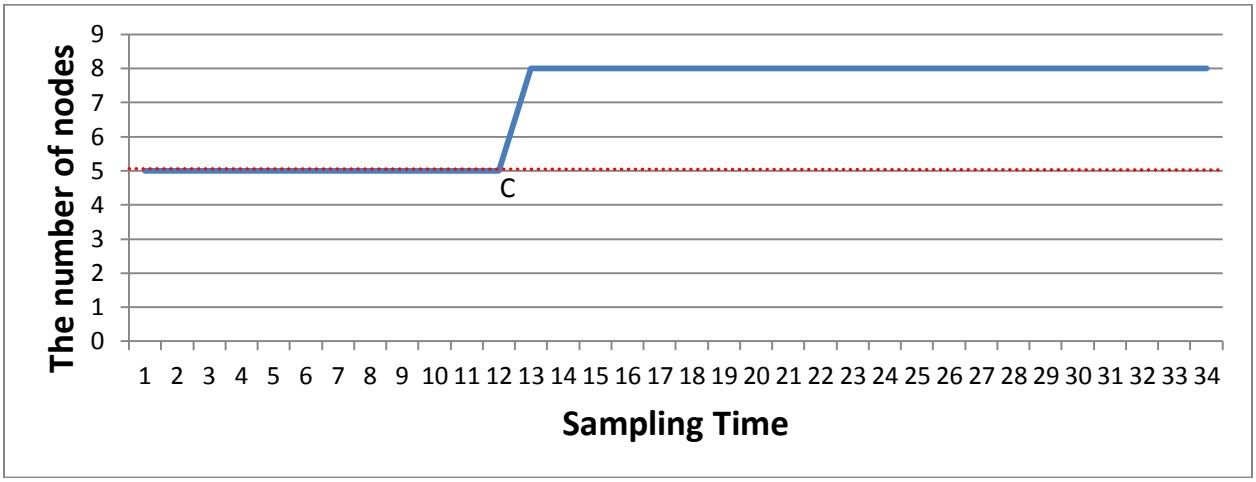

Figure 34: The changes in the number of nodes (the first experiment)

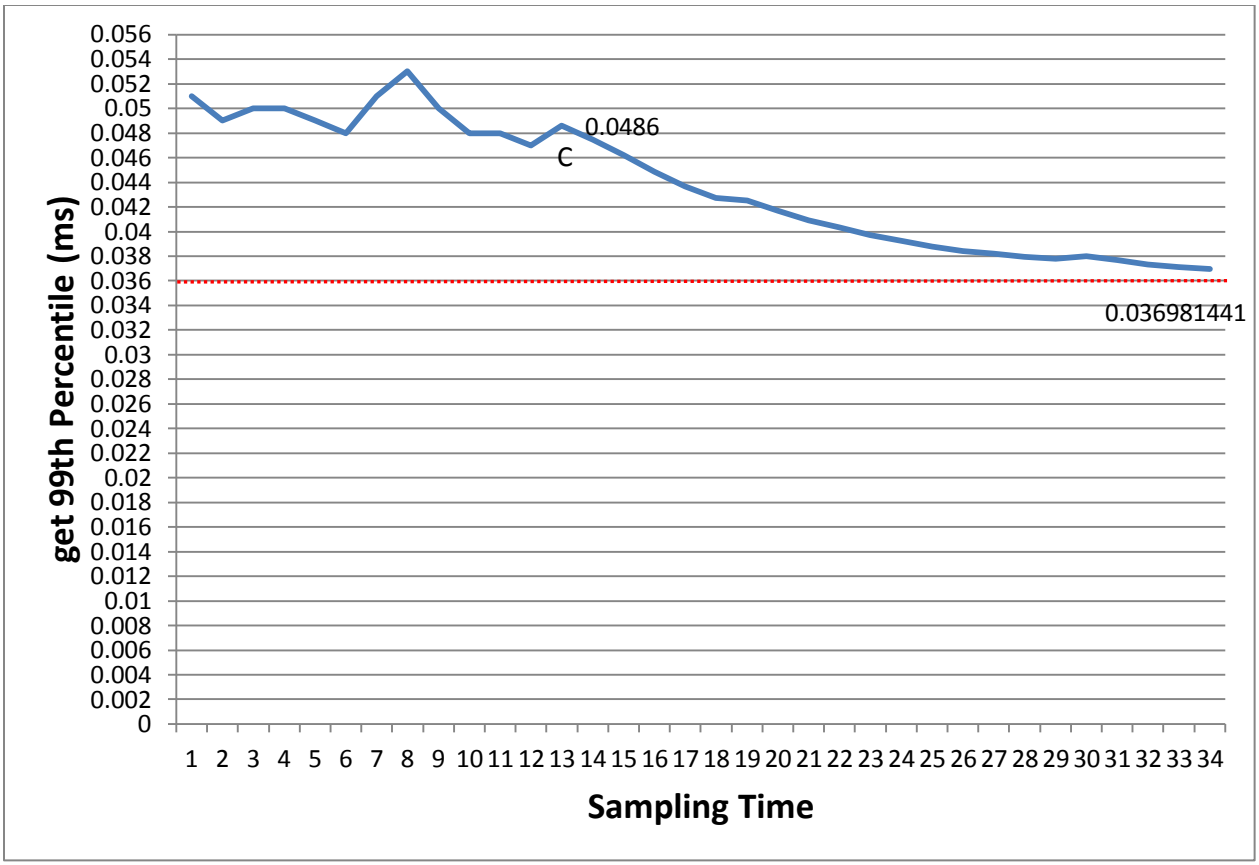

Figure 35 : The changes in get 99th percentile time(the first experiment)

4.3.2.3. Second Experiment

Figures (36) and (37) show the results of the second experiment. X axis's show the number of sampling during experiment. We can see that after warm-up period that controller starts at point ***C***, it noticed that the service time is less than SLO and removes some nodes not to waste resources and meet "pay as you go" property. The important point here is that, in this experiment we applied different gain parameters for the controller. Because, we ran several experiments with various gain parameters and found out that in case of removing nodes, these parameters lead to more precise results.

$$K_p = 12.6385024177054;$$

$$K_I = 0.107240456637031;$$

$$K_d = -501.956329851435;$$

We can consider various gain parameters in one single controller. It could be a good potential future work that in the controller implementation, different gain parameters in case of adding or removing nodes would be considered in a way that controller would detect which gain parameters would be better to apply automatically during experiment.

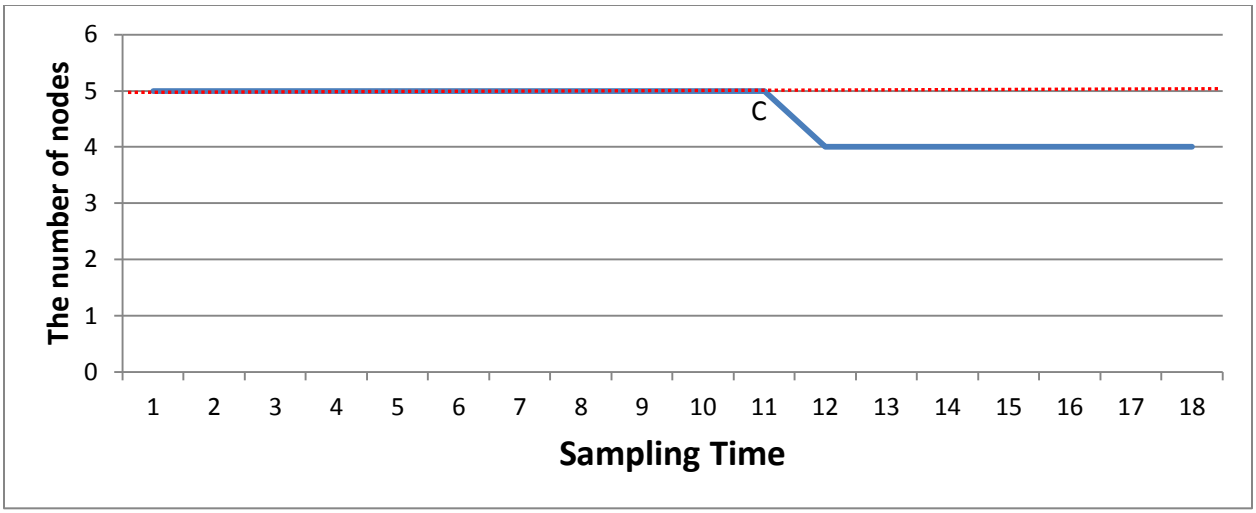

Figure 36 : The changes in the number of nodes (the second experiment)

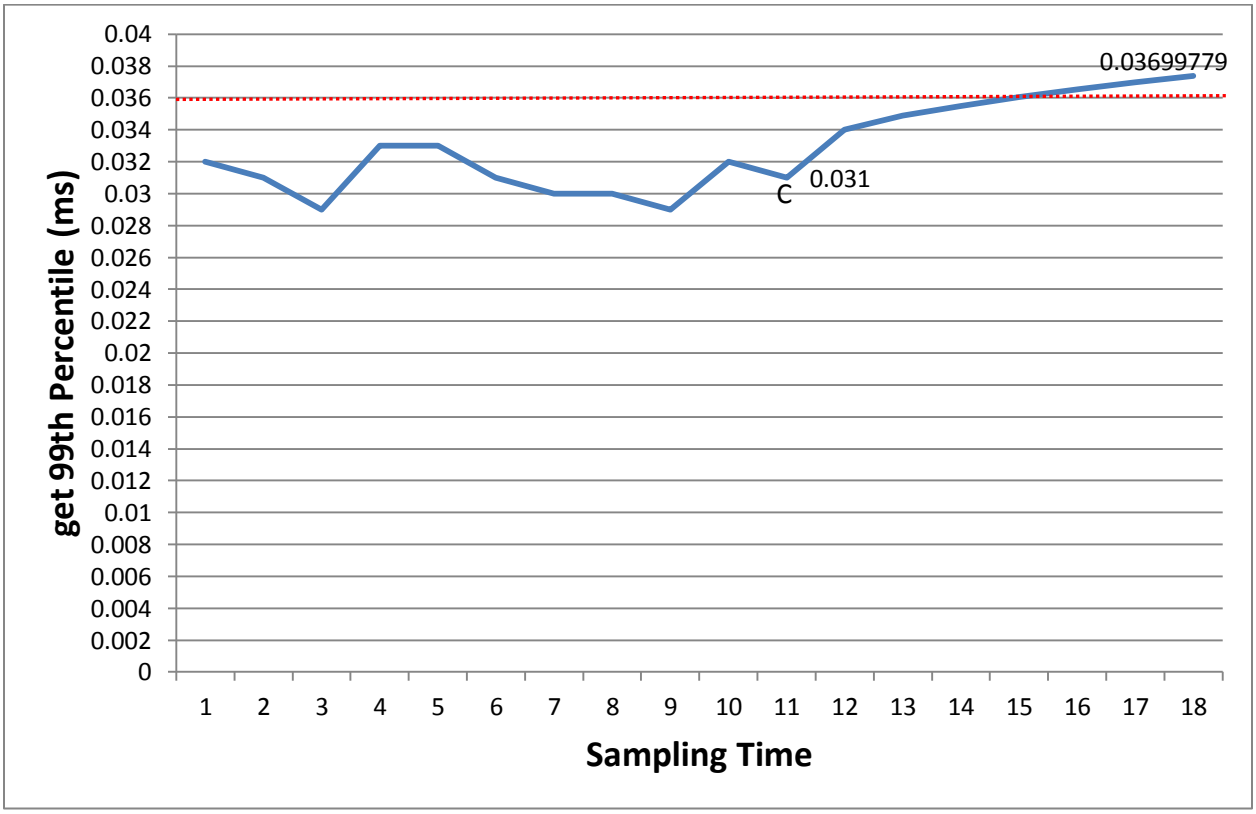

Figure 37 : The changes in get 99th percentile time(the second experiment)

The important point here is that as our system is a discrete one, the controller outputs were integer values. That is why sometimes the results were not exactly as expected. For example, consider that the controller output is 1.5. Therefore 1.5 nodes should have added which not impossible.

4.3.2.4. Third Experiment

So far, we saw that how the controller works in case of changing the *get* 99th percentile time. It means that although the load (throughput) was fixed by YCSB instances, the *get* 99th percentile timechanged. In the following we shall show that the controller works fine enough even in case of changing the load by YCSB instances.

The experiment was as following. We ran the sensor for half an hour and then the controller started at point ***C***. After about 110 minutes we increased the load. Therefore, the controller added some appropriate nodes to reach a better performance as a result. Table (16) shows the configuration of YCSB instances for this experiment. Other parameters were like previous experiments.

Table 16: The configuration of YCSB Instances for the third experiment

Warm-up Period Configuration	
Parameter	**Value**
Warm-up Period (min)	30
Throughput during warm-up (Ops/Sec)	4000
Value Size during Throughput (bytes)	1024
Benchmark (YCSB) Setup for increasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	6144

Figures (38), (39) show the results of the third experiment.

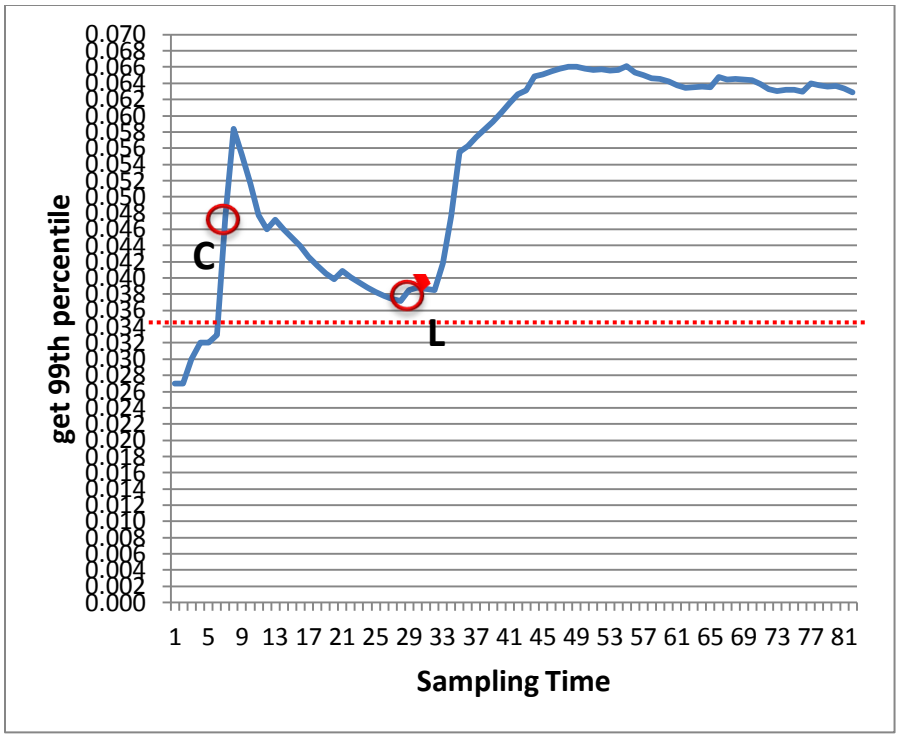

Figure 38 : The changes in get 99th percentile time(the third experiment)

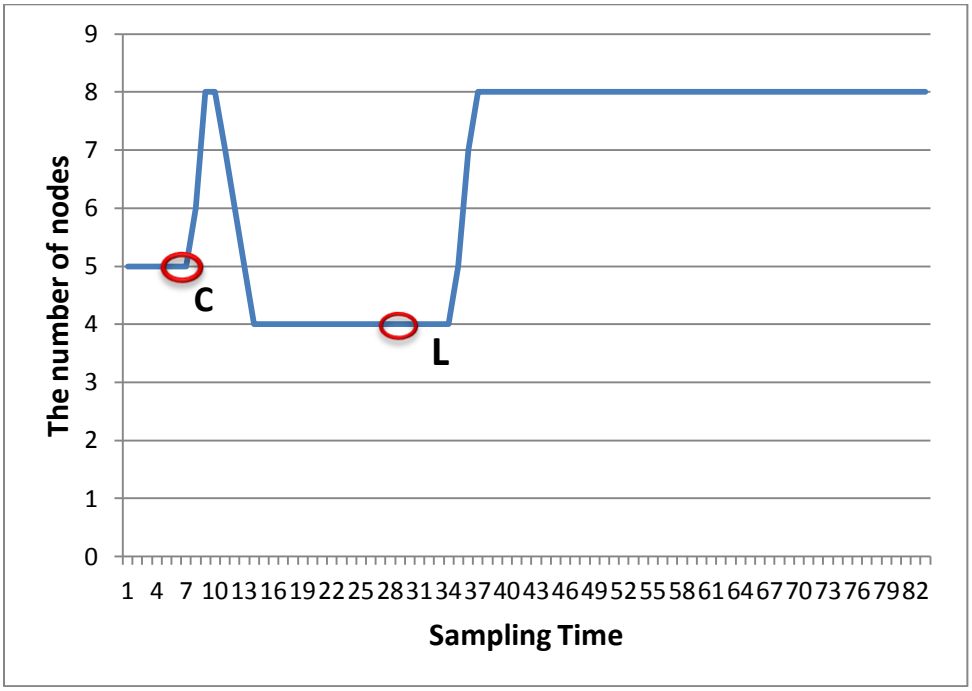

Figure 39 : The changes in the number of nodes (the third experiment)

4.3.2.5. Fourth Experiment

The experiment was as following. We ran the sensor for half an hour and then the controller started at point ***C***. After about 40 minutes we decreased the load. Therefore, the controller removes some appropriate nodes not to waste resources and meet "pay-as-you-go" property. Table (17) shows the configuration of YCSB instances for this experiment. Other parameters were like previous experiments.

Table 17 : The configuration of YCSB Instances for the fourth experiment

Warm-up Period Configuration	
Parameter	**Value**
Warm-up Period (min)	30
Throughput during warm-up (Ops/Sec)	4000
Value Size during Throughput (bytes)	1024
Benchmark (YCSB) Setup for decreasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	512

Figures (40) and (41) show the results of the fourth experiment. As we can see, at point ***C*** controller starts and after about 40 minutes we decreased the load according to Table (17) at point ***L***, the *get* 99th percentile timedecreased, although there was a little increase in the middle (which controller added some appropriate nodes to handle this increase). Therefore the controller sensed this and removes some nodes until *get* 99th percentile timereached close to SLO. Moreover we can see a good consistency between the changes in the *get* 99th percentile timeand the number of nodes.

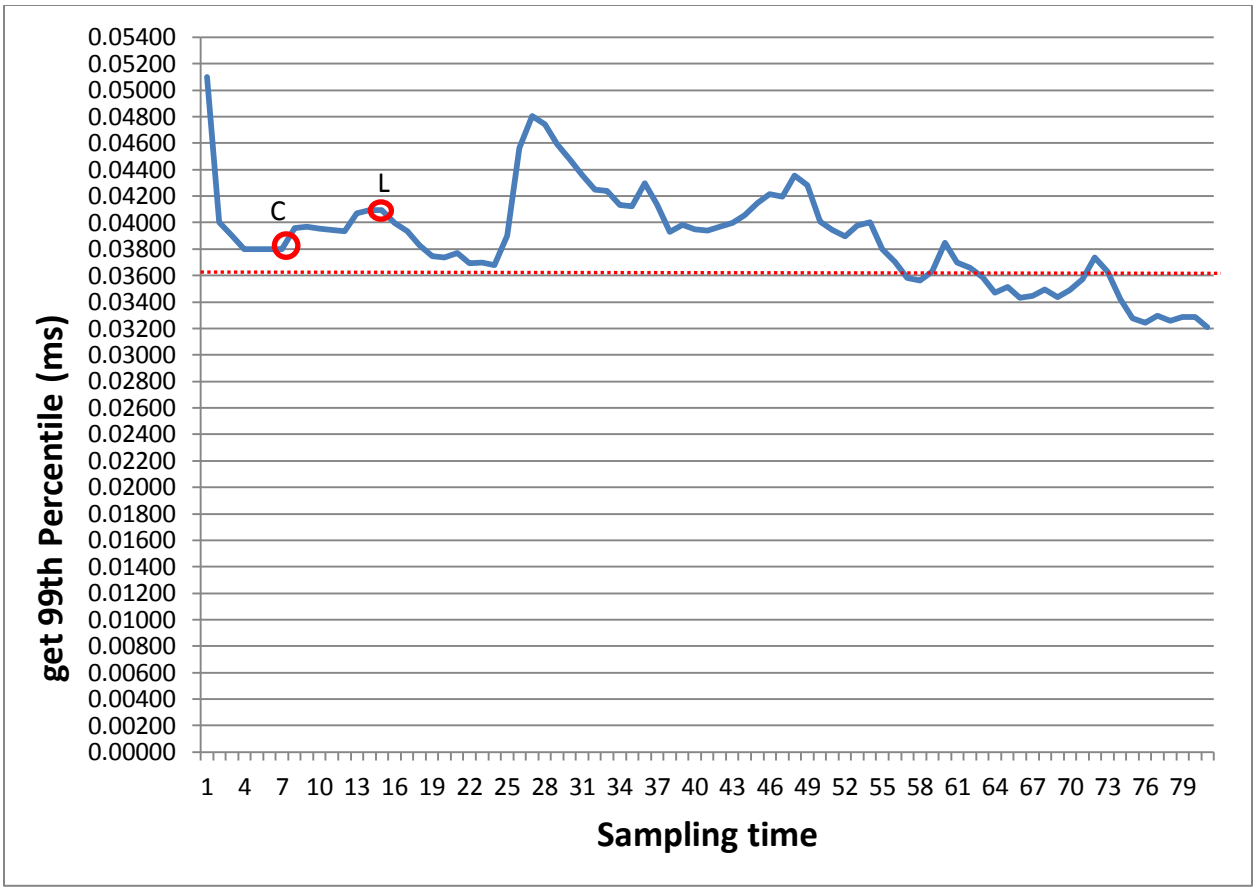

Figure 40 : The changes in get 99th percentile time(the fourth experiment)

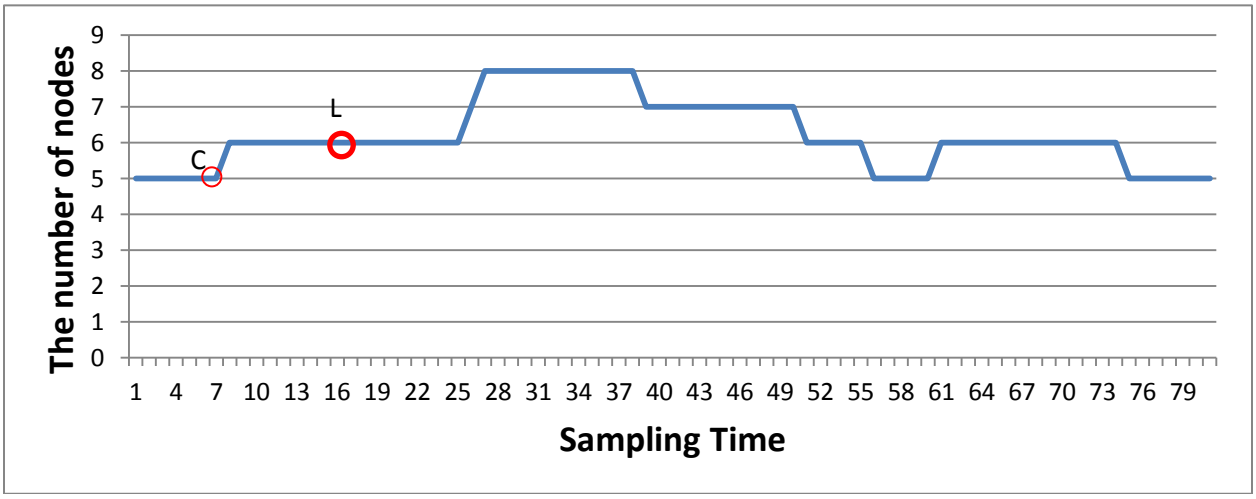

Figure 41 : The changes in the number of nodes (the fourth experiment)

4.3.2.6. Fifth Experiment

The experiment was as following. We ran the sensor for half an hour and then the controller started at point ***C***. After about 40 minutes we decreased the load. Therefore, the controller removes some appropriate nodes not to waste resources and meet “pay-as-you-go” property. Table (18) shows the configuration of YCSB instances for this experiment. Other parameters were like previous experiments.

Table 18 : The configuration of YCSB Instances for the fifth experiment

Warm-up Period Configuration	
Parameter	**Value**
Warm-up Period (min)	30
Throughput during warm-up (Ops/Sec)	4000
Value Size during Throughput (bytes)	1024
Benchmark (YCSB) Setup for decreasing Load	
Parameter	**Value**
Throughput(Ops/Sec)	500
Value size (bytes)	512

Figures (42) and (43) show the results of the fourth experiment.

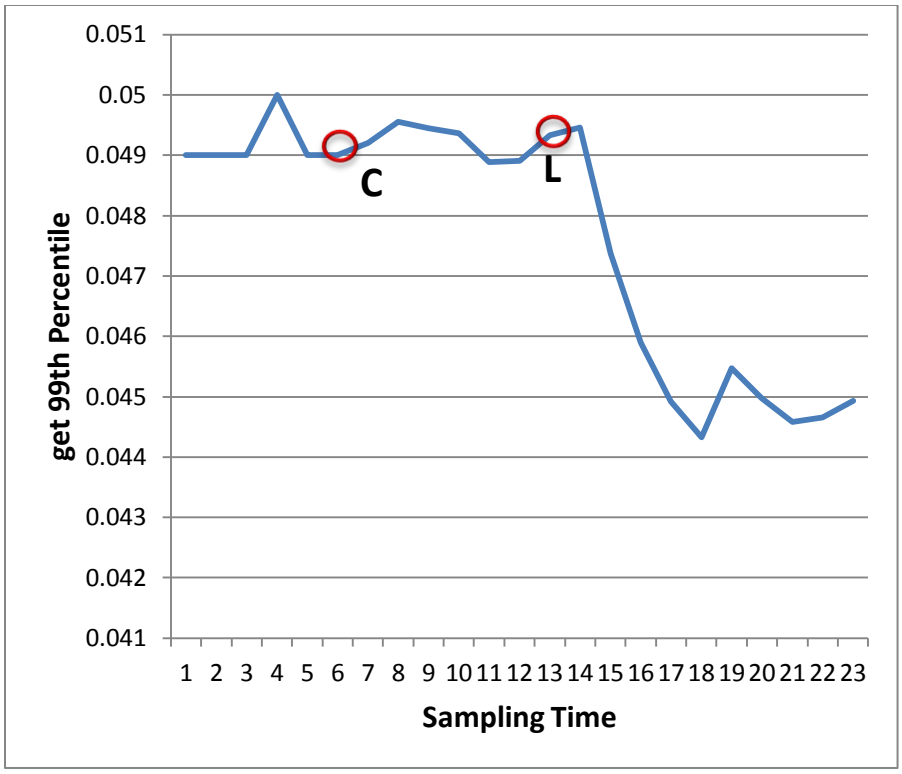

Figure 42 : The changes in get 99th percentile time(the fifthe experiment)

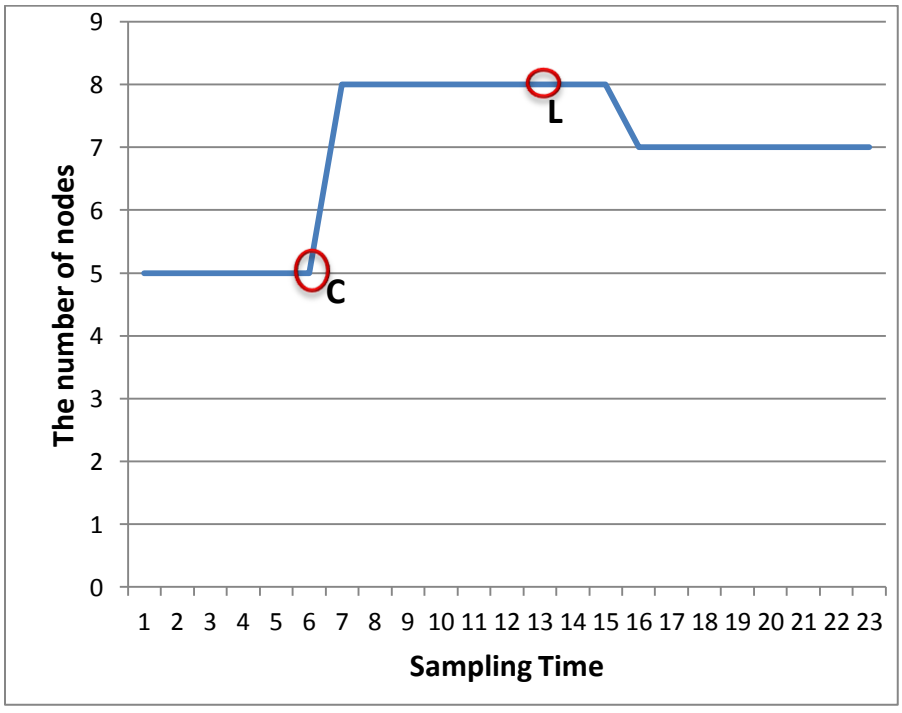

Figure 43 : The changes in the number of nodes (the fifth experiment)

Another important point in our experiments is that, as we applied a filter in our framework, the controller sees the filtered values and uses them. That is why that the changes in the number of loads were done with a little delay; but we could see smoother outputs with less spikes instead.

4.4. Summary

In this chapter we began with the continuing of the system identification with gathering input and output data to model the system. Then, we ran some simulation to show verify the controller. We saw that the designed controller is precise enough to add or remove nodes based on the incoming *get* 99^{th} percentile times. Moreover, we ran some other experiments on Voldemort using YCSB and observed that controller adds some appropriate nodes based on its parameters if *get* 99^{th} percentile timeis more than SLO and removes nodes if it is less than SLO.

Chapter 5

5. Conclusion and Future Work

In this chapter we are going to summarize and conclude what we have done in this thesis. Moreover, we will present some ideas for further work on the topic of the thesis.

5.1. Summary and Conclusions

In the first chapter, we briefly described the problem. We discussed that because of the importance of resource management in the Cloud environment, Elastic computing has become a controversial topic in nowadays. Our goal was designing a controller to monitor the load in a distributed storage called Voldemort. This controller requested for Adding/Removing nodes to control the performance of the system considering predefined SLO as well as “pay-as-you-go” property.

In this section we will show the summary of what we have done in each chapter:

- **Studying the Cloud computing**: in the background chapter, we explained the concept of the Cloud, Cloud computing and other issues in this area. We covered the different features of the Cloud computing and its advantages.
- **Studying Distributes Storage Systems (DSSs**): In the background chapter, we started with describing some various types of distributed storages. We discussed about the architecture, data model and other aspects of each. We finalized this section with a comparison of these storages from architecture, consistency and other aspects and then we came up with the motivation for choosing the Voldemort.
- **Studying Elastic computing and Autonomic computing:** In the background chapter, we discussed about the concept of elasticity in Cloud environment, Service level Agreement and its components as well as Autonomic Computing and its properties.
- **Studying the Control theory and system identification:** in the background chapter, we discussed about the concept of control theory, the objective of using the controllers,

and different types of controllers. We also presented the concept of system identification and two major approaches to identify a system.

- **Studying Voldemort**: In the background chapter, we explained different aspects of Voldemort such as configuration, design, architecture and some of its features such as versioning, failure detector, rebalancing and data partitioning.
- **Studying YCSB**: In the background chapter, we presented the architecture of YCSB as a framework for benchmarking and other features such as distributions which has been considered.
- **Related Works:** In the background chapter, we also discussed about important related works. We presented the architecture of the controller and the method of the implementation and evaluation of the system.
- **Design and Implementation of controlling framework architecture**: In chapter 3, we firstly covered the architecture of the elastic controlling framework and its implementation. Then we went through system identification, system modeling and the details of control design.
- **Experiment and evaluation**: In chapter 4, we continued system identification by running several experiments and collecting input/output data. Finally we ran some experiments using the controller, YCSB on Voldemort and evaluate the results.

We designed and implemented a controller that monitored the load in the system and then requested for grab/release nodes based on the algorithm achieved during the designing the controller. It was clear that, considering a predefined SLO, the controller could sense that the load was increasing and added some resource based on its parameters. Similarly, it could sense that the load was decreasing and it removed some nodes (resources).

5.2. Future works

In this section we shall introduce some ideas as some extensions to this thesis which may come to attention:

5.2.1. Distributed Controller

One can design a distributed controller instead of a central controller. In our experiments, the cluster had only 8 nodes; therefore a central controller was enough. But in a huge cluster which includes millions of nodes a central controller cannot be efficient. Some advantages of a distributed controller are:

- No single point failure
- **Performance**: More speed in comparison with the central one
- **Scalability**: In the case of adding more resources to the cluster, we can add more controllers to the system to manage them. This is similar for decreasing the size of the cluster.

Indeed it is obvious that a distributed controller needs more maintenance such as consistency, security and mask failures issues.

5.2.2. Decreasing the effect of rebalancing

In our experiments, there was limited data in the database in comparison with real world scenarios. In addition, we did not delete the data from the current nodes when we wanted to add some nodes remove some nodes. Consider the case that Voldemort wants to add one or more node to the cluster and needs to move data from some existing nodes to the new nodes. This process could take hours. Future work can focus on the decreasing the effect of rebalancing on the performance of the system.

5.2.3. Handling the load spikes

In our experiments, we did not consider the spikes in the system. For example, we considered the difference of the load during the day and the night which is more in the first case. Then we added some resources during the day. However, in some special cases, there could be some load spikes, meaning that load suddenly jumps up due to a specific event. Future work could be on how to handle these load spikes.

5.2.4. CPU usage as a touch point

In this thesis, Average get 99^{th} percentile timeused as a touch point. In other words, we measured it to see if the load has been increased or decreased. We could consider (average) CPU usage of each machine that nodes are running on them. Future work can focus on working on this part of the problem.

5.3. Summary

In this chapter we discussed about the summary of what we have done during the master thesis and what result we got. We showed that we have met our objectives. Then we finalized with some possible options for future work.

6. References

[1] Robert L. Grossman, 2010, Compute and Storage Clouds Using Wide Area High Performance Networks, http://arxiv.org/PS_cache/arxiv/pdf/0808/0808.1802v1.pdf, Last access on 21 January, 2012

[2] Daniel J. Abadi, 2009, Data Management in the Cloud: Limitations and Opportunities, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.200&rep=rep1&type=pdf, Last access on 21 January, 2012

[3] Dr Ashish Rastogi, 2010, A Model based Approach to Implement Cloud Computing in E-Governance, http://www.ijcaonline.org/volume9/number7/pxc3871888.pdf, Last access on 21 January, 2012

[4] Mehmet Sahinoglu, 2010, Cloud Computing,

http://onlinelibrary.wiley.com/doi/10.1002/wics.139/pdf, Last access on 21 January, 2012

[5] Michael Armbrust, 2010, A View of Cloud Computing, http://cacm.acm.org/magazines/2010/4/81493-a-view-of-Cloud-computing/fulltext, Last access on 21 January, 2012

[6] Paramvir Bahl, 2006, Discovering Dependencies for Network Management, http://research.microsoft.com/en-us/um/people/ranveer/docs/and.pdf , Last access on 21 January, 2012

[7] Krishna Nadiminti, 2006, Distributed Systems and Recent Innovations: Challenges and Benefits, http://www.buyya.com/papers/InfoNet-Article06.pdf, Last access on 21 January, 2012

[8] Alessandro Duminuco, 2007, Proactive Replication in Distributed Storage Systems Using Machine Availability Estimation,

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.3947&rep=rep1&type=pdf, Last access on 21 January, 2012

[9] Avinash Lakshman, 2010, Cassandra - A Decentralized Structured Storage System, http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf, Last access on 21 January, 2012

[10] Cassandra Wiki, http://wiki.apache.org/cassandra/ArchitectureOverview, Last access on 21 January, 2012

[11] Hbase Home, http://hbase.apache.org, Last access on 21 January, 2012

[12] Building Voldemort read-only stores with Hadoop, http://sna-projects.com/blog/2009/06/voldemort-and-hadoop, Last access on 21 January, 2012

[13] Jay Kreps, 2009, Project Voldemort,

http://qconsf.com/dl/qcon-sanfran-2009/slides/JayKreps_ProjectVoldemortScalingSimpleStorageAtLinkedIn.pdf, Last access on 21 January, 2012

[14] Werner Vogels' weblog on building scalable and robust distributed systems, http://www.allthingsdistributed.com/2010/02/strong_consistency_simpledb.html, Last access on 21 January, 2012

[15] Hadoop vs. Voldemort, http://www.slideshare.net/bhupeshbansal/hadoop-user-group-jan2010, Last access on 21 January, 2012

[16] Brian F. Cooper, 2010, Benchmarking Cloud Serving Systems with YCSB, research.yahoo.com/files/ycsb.pdf, Last access on 21 January, 2012

[17] Voldemort Performance Tool, https://github.com/voldemort/voldemort/wiki/performance-tool, Last access on 21 January, 2012

[18] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/ , Last access on 21 January, 2012

[19] Petr Sobeslavsky, 2011,

Elasticity in Cloud Computing, www.sobeslavsky.net/files/elasticity.pdf, Last access on 21 January, 2012

[20] Linlin Wu, 2010, Service Level Agreement (SLA) in Utility Computing Systems, http://www.Cloudbus.org/reports/SLA-UtilityComputing2010.pdf, Last access on 21 January, 2012

[21] Jeffrey O. Kephart, 2003, The Vision of Autonomic Computing,

http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf,

, Last access on 21 January, 2012

[22] Lennart Ljung, 2010, Perspectives on System Identification, http://www.control.isy.liu.se/~ljung/seoul2dvinew/plenary2.pdf, Last access on 21 January, 2012

[23] Rik Pintelon, 2001, System identification: a frequency domain approach, ISBN-10: 0780360001

[24] Hilbert J. Kappen, 2008, An introduction to stochastic control theory, path integrals and reinforcement learning, http://www.snn.ru.nl/~bertk/kappen_granada2006.pdf, Last access on 21 January, 2012

[25] Tarek Abdelzaher, 2008, http://www.cs.wustl.edu/~lu/papers/sigmetrics08_control.pdf,

Introduction to Control Theory And Its Application to Computing Systems, Last access on 21 January, 2012

[26] Guillermo J. Costa, 2011, Tuning a PID Controller, http://www.powertransmission.com/issues/0411/pid.pdf, Last access on 21 January, 2012

[27] Harold C. Lim, 2010, Automated control for elastic storage, http://dl.acm.org/citation.cfm?id=1809051&bnc=1, Last access on 21 January, 2012

[28] Patrick Martin, 2011, Autonomic Management of Elastic Services in the Cloud

[29] Katsuhiko Ogata - Fifth Edition, 2009, Modern Control Engineering, ISBN-10: 0136156738

[30] IBM Corporation, 2006, An architectural blueprint for autonomic computing, http://www.ginkgo-networks.com/IMG/pdf/AC_Blueprint_White_Paper_V7.pdf, Last access on 21 January, 2012

[31] Apache commons, http://commons.apache.org/math/, Last access on 21 January, 2012

[32] J. L. Pon, 2005, Emerging Actuator Technologies: A Micromechatronic Approach, http://media.wiley.com/product_data/excerpt/75/04700919/0470091975.pdf, Last access on 21 January, 2012

[33] Matlab Help

[34] Carstoiu, D, 2010, Hadoop Hbase-0.20.2 performance evaluation

[35] mongoDB, http://www.mongodb.org/, Last access on 21 January, 2012

[36] Kristina Chodorow, 2010, MongoDB: The Definitive Guide, ISBN: 1449381561

[37] Voldemort Homepage, http://project-voldemort.com/, Last access on 21 January, 2012

TRITA-ICT-EX-2012:20

www.kth.se