Master of Science in Computer Science Engineering
February 2018

Performance Evaluation of Cassandra Scalability on Amazon EC2

Siddhartha Srinadhuni

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Siddhartha Srinadhuni
E-mail: sisr16@student.bth.se

University advisor:
Dr. Emiliano Casalicchio
Department of Computer Science and Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Context In the fields of communication systems and computer science, Infrastructure as a Service consists of building blocks for cloud computing and to provide robust network features. AWS is one such infrastructure as a service which provides several services out of which Elastic Cloud Compute (EC2) is used to deploy virtual machines across several data centers and provides fault tolerant storage for applications across the cloud.

Apache Cassandra is one of the many NoSQL databases which provides fault tolerance and elasticity across the servers. It has a ring structure which helps the communication effective between the nodes in a cluster. Cassandra is robust which means that there will not be a down-time when adding new Cassandra nodes to the existing cluster.

Objectives. In this study quantifying the latency in adding Cassandra nodes to the Amazon EC2 instances and assessing the impact of Replication factors (RF) and Consistency Levels (CL) on autoscaling have been put forth.

Methods. Primarily a literature review is conducted on how the experiment with the above-mentioned constraints can be carried out. Further an experimentation is conducted to address the latency and the effects of autoscaling. A 3-node Cassandra cluster runs on Amazon EC2 with Ubuntu 14.04 LTS as the operating system. A threshold value is identified for each Cassandra specific configuration and is scaled over to five nodes on AWS utilizing the benchmarking tool, Cassandra stress tool. This procedure is repeated for a 5-node Cassandra cluster and each of the configurations with a mixed workload of equal reads and writes.

Results. Latency has been identified in adding Cassandra nodes on Amazon EC2 instances and the impacts of replication factors and consistency levels on autoscaling have been quantified.

Conclusions. It is concluded that there is a decrease in latency after autoscaling for all the configurations of Cassandra and changing the replication factors and consistency levels have also resulted in performance change of Cassandra.

Keywords: Cassandra, Amazon EC2, Performance Evaluation, Scalability, Cloud Benchmarking, Data Scalability, Infrastructure as a service

ACKNOWLEDGEMENTS

I would like to thank and extend my warm gratitude to Dr. Emiliano Casalicchio without whose guidance, mentorship and support, this research would have been cumbersome to perform.

I would like to remember my grandfather late Talluri Raja Rama Mohan Rao for his love and constant support that has driven me to have a curious mind. I would also like to thank my seniors Sharath Reddy Abbireddy, Navneet Reddy Chamala and Gopi Krishna Devulapally who have helped me with the technical intricacies within this research and for their unconditional support.

I would like to thank Neeraj Reddy Avutu and Ravi Shankar Kondoju for their inputs within data analysis.

Finally, I would like to thank my parents Kishan Rao Srinadhuni and Sarvani Talluri for their constant guidance and motivation and most of all, planning my career in Computer Science.

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

The focus of this research is to analyze and quantify the scalability of Cassandra running on Amazon Web Services (AWS) platform. Specifically, this research is aimed towards measuring the latency in adding the Cassandra nodes through AWS, when a replication factor and consistency levels are to be maintained. As a part of this research, experimentation is used as a research method to collect the results. Primarily, a Cassandra cluster is subjected to stress via the tools mentioned in this document, through which we have measured the CPU utilization of the Cassandra cluster. These CPU utilization values are used to auto-scale the Cassandra cluster on Elastic Compute Cloud (EC2) instances. Based on the results obtained, latency of the Cassandra nodes adding to the existing cluster is measured and thus the performance of Cassandra scalability has been evaluated. Furthermore, to investigate the impact of Replication Factors (RF) and Consistency Levels (CL) of Cassandra on auto-scaling, the experiment is repeated with the given parameters and the latencies are quantified.

1.1 Contribution

The prime contribution of this thesis is to quantify the latency in adding the Cassandra nodes on AWS platform and the effects of RF, CL on auto-scaling. It is observed from the readings that the nodes take considerable amount time to start and run. It is investigable to quantify this latency and to quantify the latency with RF and CL as parameters. This model gives an idea of how Cassandra scales and the scalability has been evaluated by considering different metrics.

1.2 Thesis Outline

This section describes how this research document is structured. The chapter that follows this section explains briefly about the Background of Cloud Computing, AWS, Cassandra and the intricate details of these technologies which have been a part of this research. The Chapter following it is Related work and this chapter describes the research done so far with relevance to this research, what additional contribution can be made and in which areas does this research fit best. Furthermore, the research gap and the research questions are formulated in this section. Following it is the research methodology which explains how the experiment has been carried out and what resources does the experiment demand and so forth. Results and analysis of the experimentation follow then. Discussions, Conclusions and Future scope of this research have been explained rather in an elaborative way towards the end of this document.

1.3 Background

1.3.1 Cloud Computing

The word ‘cloud’ was first used to describe the business model of providing services across the internet in 2006. Cloud Computing eases the way of accessing the data and (or) applications. Amazon Web Services (AWS) is one such kind of Cloud Computing provider which provides vast services apart from providing confidentiality, integrity and availability of the customer’s data [1], [2].

1.3.2 Amazon EC2 and service modules

This chapter explains how different services provided by AWS EC2 infrastructure work and how they are configured in order to support autoscaling. For this thesis study, an amazon free tier account is used so that there is a scope of using a 100$ worth services on AWS for free. The free tier eligibility is only for *t2.micro* type instances which are used for the initial phases

of experimentation where there is no deduction of the credits. The *t2.micro* instance offers 1 GiB of memory and costs 6 credits per hour. The main experiment, however has been done on *t2.medium* instances which offer 4GiB of memory and costs about 24 credits per hour.

EC2 instance configuration type	T2.micro	T2.medium
vCPU	1	2
Memory (GiB)	1	4
Cost	6 credits/hour	24 credits/hour
Operating System	Ubuntu 14.04 LTS	Ubuntu 14.04 LTS

Table 1-1: Types of EC2 instances' configurations [3]

1.3.2.1 Virtual Private Cloud(VPC) and Subnets

VPC allows the user to create a virtual network on AWS that uses intends to define spanning across different Availability Zones (A-Zs) in the region. This VPC acts as the networking layer for AWS and is very much similar to the network in a conventional data center so that the user does not have to get acquainted with new nuances to operate. VPC also includes the perks of using the scalable infrastructure and is isolated from other virtual networks in the AWS Cloud. Configuring the VPC includes modifying the IP range, configuring the route tables and security settings[4].

Subnets are defined as the range of IP addresses in the VPC. This enables the user to launch his EC2 instances into a specified subnet in the VPC and the subnet that the user is associated with a route table which describes the allowed routes for outgoing traffic [4].

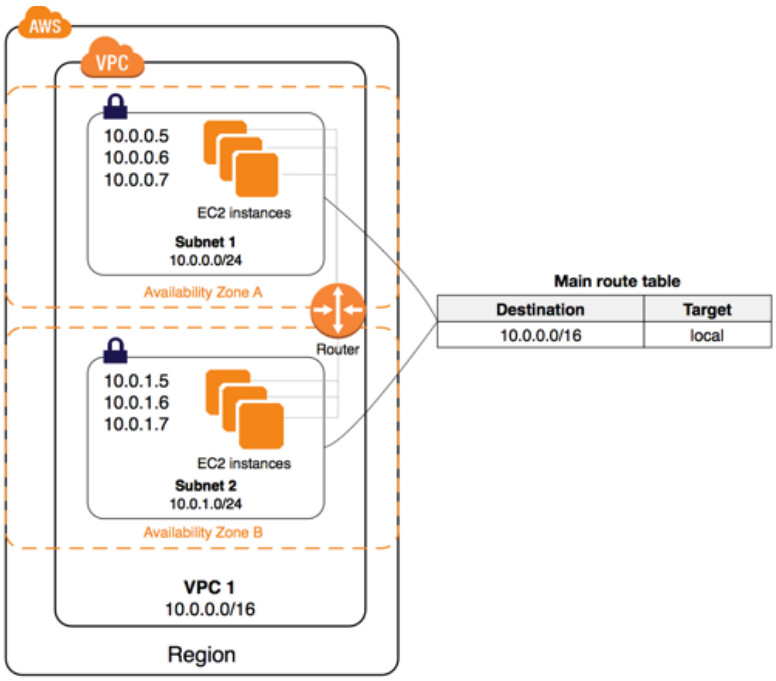

Figure 1-1. Amazon Virtual Private Cloud [4]

1.3.2.2 Launch Configuration

Autoscaling gives the user an option to the user to inherit the attributes from an existing EC2 machine to Launch Configurations. Launch Configuration is like a set of sample document that has few details already in place. It acts like a proper template that the Autoscaling Groups uses to spin up or launch the instances[5].

1.3.2.3 Security Groups

Security Groups in AWS acts as the firewall that monitors and controls the outbound and incoming traffic. Launching an instance from the scratch or launching an instance through launch configuration would prompt the user to enable rules for one or more security groups. The rules for inbound traffic can be TCP, UDP, ICMP or All traffic which means that the filtration of is done based on the type of the rule that the user specifies depending upon the traffic type that she intends to allow [6].

1.3.2.4 Scaling policy

Autoscaling in AWS is implemented by two scaling policies. They are simple scaling and step-scaling policies. These scaling policies are defined in a user interactive space wherein the user specifies when the instances should scale, how many instances should be deployed depending upon the metrics user finds suitable for scaling. Further, after the scaling policy is created, this policy awaits the scaling activity and when the metric specified has reached the defined value, the alarm breaches its threshold value and launches the mentioned number of instances into the Autoscaling group where these newly generated instances bootstraps to the existing instances. The metrics provided for the alarm to breach, as a part of this research is the mean CPU Utilization [7].

1.3.2.5 Auto-scaling Groups(ASG)

A collection of EC2 instances when they have analogous characteristics and considered as a logical grouping for scaling and management is called an Autoscaling Group. The prime use-case of this ASG is to increase the number of instances to enhance the performance of the deployed application. In ASG, the user can specify the number of instances that are desired and the number of instances that are terminated. Health checks on the instances are performed in regular intervals to ensure the resiliency and if there is an unhealthy state of an instance, the ASG will terminate it and launches another instance to replace it [8].

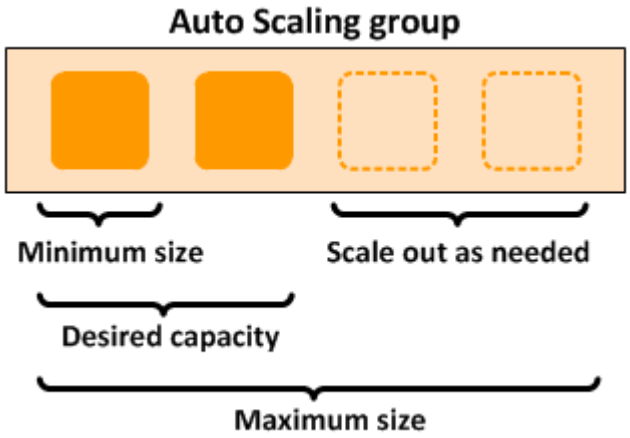

Figure 1-2. Autoscaling Mechanism [8]

1.3.3 Cassandra

Cassandra has been the prime System Under Test in this research and it has been selected for this research because of the following rationales:

- Cassandra is one of the major NoSQL databases provides high data reliability and availability unlike the traditional databases.
- It is also used for horizontal, vertical scaling and operational ease across different data centers.

- Cassandra is a fault tolerant system which means that there is no single point failure and can handle large amounts of data with ease unlike the relational databases [9].
- It is scalable which means that even if additional data is introduced to the system, Cassandra performance does not show any downtime.
- The scope in research associated with Cassandra is wide since there are different configurations to explore such as Replication Factors, SSTables, Consistency Levels and so on [10].
- Further, every node in the Cassandra cluster is identical therefore bootstrapping algorithms work efficiently in joining the newly generated nodes and starts gossiping as soon as they join the cluster[11].

1.3.3.1 Consistency Levels (CL) and Replication Factors (RF)

There are three main levels of consistency which Cassandra offers, both for READ and WRITE. They are ONE, QUORUM and ALL. When only one replica node in the cluster is required to reply to the query which means that only it contains the dataset which is required to answer the query, then it is said that the Consistency level is set to ONE. When D is the replication factor, Consistency level of QUORUM implies $Q=D2+1$ replica nodes are available to reply to the query and consistency level ALL means that all the replicas are available [12].

Replication factor is defined as the number of replicas of Cassandra that are viable across the nodes present in the cluster. For example, if the RF is set to 2, the data gets copied into 2 nodes in the cluster. Replication factor is limited to the number of nodes in the cluster. Further, a replication strategy is used to determine the nodes where the replicas are placed [13]. Since there is only one data center involved in this experiment, a *SimpleStrategy* is used. On the other hand, *NetworkStrategy* is used to deploy Cassandra on multiple Data centers. The replication factor acts as an attribute to keyspace which is a peripheral container for data in Cassandra. The following diagram illustrated how a keyspace looks.

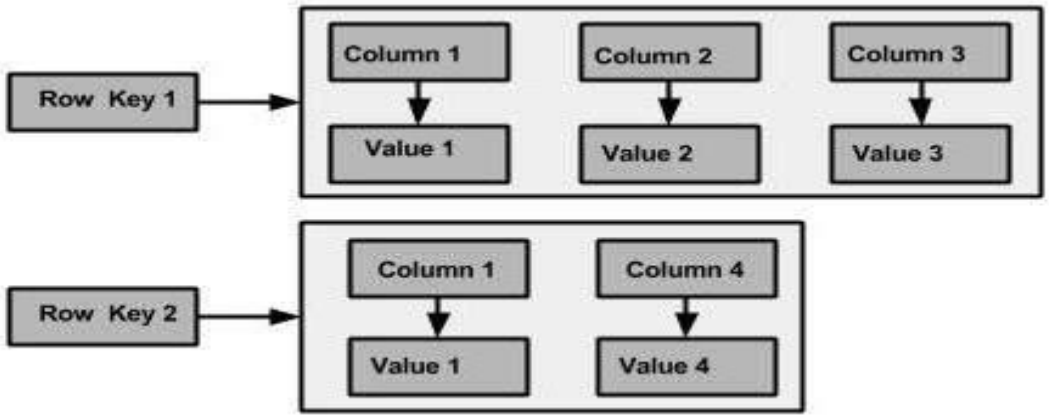

Figure 1-3. Keyspaces in Cassandra

1.3.3.2 Cassandra Query Language

Cassandra Query Language is Cassandra specific query language which is used to create, manage, manipulate and terminate data in Cassandra. In this experiment, CQL has been used to create and drop keyspaces and to define replication factors. A keyspace in Cassandra defines the data replication on nodes. For example, consider the following query [14].

```
CREATE KEYSPACE "keyspace1" WITH replication = {
  'class': 'SimpleStrategy',
  'replication_factor': '3' };
```

This creates a keyspace with the name keyspace1 with a replication factor of 3 and with SimpleStrategy as the replication strategy.

1.3.3.3 Bootstrapping in Cassandra

A random token is chosen when a Cassandra node is started for the very first time. The token information is then gossiped among all the nodes present in the cluster. Therefore, the number time taken for bootstrapping is relatively higher in a cluster with a higher number of nodes. This is how all the nodes in the cluster know about their respective positions in the cluster [11].

1.3.4 Cassandra Stress tool

Cassandra stress tool is a benchmarking tool designed to test a Cassandra cluster. This Java-based stress tool is used to for basic benchmarking and to load test the Cassandra cluster. This tool makes the user understand how the database under test scales and helps determine the capacity of Cassandra in this research. Number of reads and writes can be defined in the scenario to load test the Cassandra cluster. In this research, the number of reads and writes is equal which means that the workload has been the same in both the cases. When this tool is used, tables along with the consistency level specified are created on the target cluster within a newly instantiated keyspace. To measure the metrics such as latency and CPU Utilization, these tables are used [15].

1.3.5 Dstat Tool

Dstat Tool is a monitoring tool in Linux which is used to perform statistical analysis of several system resources like memory, CPU and network in real time. Dstat also supports writing the logs to multiple file formats like .csv and .log. A major advantage of using Dstat is that it measures CPU utilization and disk utilization simultaneously [16].

2 RELATED WORK

This chapter describes the research that has been done so far in the problem domain of this research. Various literature has been scrutinized to understand the state-of-the art of experimentation in the field of benchmarking NoSQL databases and assessing their performance. This chapter has been categorized into two subsections where the first subsection describes the research which has been done in benchmarking Cassandra and the second subsection describes the research done so far AWS EC2 infrastructure.

2.1 Cassandra

The problem of quantifying the latency in adding Cassandra nodes through autoscaling has been covered in different fields of research including NoSQL databases, Cloud Computing, Performance evaluation to name a few. To list a few examples within this field of scaling and benchmarking Cassandra nodes or other NoSQL databases [3]–[5], [13] and [20] has compared the performance of Cassandra, Hbase with the other traditional databases with taking replication into consideration, taking latency as the metric to compare and using Yahoo Cloud Serving Benchmark (YCSB) tool. [21] by Yishan Li et al. have compared the performance of several NoSQL databases including Cassandra, MongoDB, CouchDB and so on with MySQL and have taken throughput and key-value stores as a metric to compare. On the other hand [22] has compared Cassandra, MongoDB and Riak based on the use cases with throughput and latency as the factor to compare. [18] by Veronika et al. scales Cassandra cluster with the YCSB tool and with different workloads and also concludes that scaling the number of nodes does not guarantee an improvement in the performance. Furthermore, the performance in this paper has been evaluated by comparing the execution times of different workloads and varying the number of threads. [19] by Gandini et al. claims that the replication factor in a database is a performance impacting factor and used replication factor as a parameter to be manipulated.

2.2 Amazon EC2

The research paper by Kuhlenkamp et al. [17] has compared latency of Cassandra while autoscaling it with the latencies of other such NoSQL database, HBase and have shown that there is a tradeoff between the latencies when compared but it is also said that the AWS EC2 infrastructure plays a pivotal role in terms of scaling, time taken to scale and data volumes of both databases on the infrastructure. [23] has evaluated the consistency of Cassandra clusters with different configurations on AWS EC2 infrastructure.[24] has evaluated two Cassandra clusters on two different configurations of EC2 infrastructure. Further, investigations of response time and throughput have been carried out and present a methodology for Quality of Service [QoS] aware provisioning of Cassandra clusters.

2.3 Research Gap

Cassandra is a reliable, non-relational and distributed database which has automatic replication and can manage large amounts of data spread across multiple commodity servers[11]. This means that Cassandra can duplicate its records throughout the cluster and is set by the user in control. Also, when Cassandra nodes are auto scaled there shall be no downtime or interruptions to the applications.

AWS offers IT infrastructure services to businesses in the form of webservices which provide a reliable, scalable and cost-efficient platform in the cloud. EC2 is one such web service interface from AWS which reduces the time to scale and pay for the capacity that is only used by the user. Cassandra has seen multiple research attempts within evaluating the scaling,

comparing the performance with its peer non-relational databases for Key Performance Indicators and [12], [21], [18], [19]. Research has also been performed to understand the dynamics of Autoscaling on IaaS cloud [25] and dynamics of scaling Cassandra nodes on AWS [17]. This leads to a research gap to measure the latency, a key performance metric to evaluate scalability while autoscaling on AWS. This research will quantify the latency of adding Cassandra nodes to AWS and shall show the effects of autoscaling on the Cassandra cluster. Further, a research gap leading to identify the effects of autoscaling on different configurations of Cassandra by tweaking the RF and CL will also come in light. These underlying issues of how to measure the latency in adding Cassandra nodes and to measure it along with the effects of autoscaling shown on different configurations of Cassandra have been answered through this research.

2.3.1 Aims and Objectives

2.3.1.1 Aim:

The main aim of this research is to analyze and to quantify the scalability of Cassandra when it runs on Amazon EC2 instances. For this, the Cassandra nodes are horizontally scaled which means that adding or removing virtual nodes at runtime, e.g. adding Cassandra virtual nodes to the Cassandra Virtual Data Centers (VDC) [12]. In particular, this research is aimed at measuring the latency in adding Cassandra nodes. To fulfill the aim and to define the key performance indicators of Cassandra scalability, the following are considered as objectives.

2.3.1.2 Objectives:

- To narrow down the key performance indicators of scalability of Cassandra, literature should be keenly studied.
- To form a three-node and a five node Cassandra cluster so as to measure the performance indicators, CPU utilization and Latency, on EC2 instances when the read and write operations are performed using stress tool.
- Understanding the replication factors and consistency levels in Cassandra and identifying their effects on auto-scaling of Cassandra.
- Recording the CPU utilization and the time values using the tools available.
- Measuring the latency in adding Cassandra nodes when the Cassandra cluster auto-scales on AWS platform.
- Comparing the CPU utilization and the latencies when different replication factors and consistency levels are maintained on the cluster thereby determining the impact of RF and CL on auto-scaling.

2.3.2 Research Questions

RQ1) How to measure and what is the latency in adding Cassandra nodes to Amazon EC2 instances?

Motivation: The goal of this research question is to address the underlying problem of latency, and to determine it when adding Cassandra nodes to EC2 instances. The latency can be found out when the cluster auto-scales and the new nodes start to communicate with the cluster. There is be an observed change in the CPU utilization when the auto-scaled nodes start communicating with the cluster corresponding to each of the different Cassandra configurations.

RQ2) How do the replication factors and consistency level impact the auto scaling of Cassandra Virtual data centers?

Motivation: This research question is framed to identify and assess the impact level on Cassandra virtual data centers by different consistency levels and replication factors. Different

RFs and CLs consume CPU at different rates and hence the CPU utilization are prone to change with the change in RF and CL. Therefore, the auto-scaling is also affected. Quantifying the latency in this case of changing the Cassandra configurations in a 3 and a 5node cluster, is the goal of this research question. It can be measured by creating keyspaces, repeating the experiment and measuring the latency through the benchmarking tool.

3 METHODOLOGY

This chapter describes the different methods that are used as a part of answering the research questions that are considered in this research. This research is aimed to study the performance of Cassandra scalability implemented on Amazon EC2 platform when benchmarked by Cassandra-stress tool. Primarily, different performance factors are identified by conducting a literature review. There is more literature to be reviewed to study the state of the art in experimentation, providing a strong theoretical foundation to the proposed study and substantiating the need for further experimentation [26].

3.1 Literature Review

For any research to be conducted, appropriate literature has to be studied so as to diagnose, identify and formulate an idea or a concept [27]. Therefore, during the initial phases of this research, in order to formulate the idea in this context, a literature review is conducted. A literature review is carried out to learn two major aspects in this scenario. One of them being, the study of Cassandra and gaining knowledge about the dynamics of Cassandra, its configurations and metrics to measure the performance of Cassandra Scalability. The other aspect is to know and learn about the AWS EC2 platform and autoscaling policies. The following Inclusion/Exclusion criteria has been defined to scrutiny the literature obtained.

- Is the article in English?
- Is the article published between 2005-2017?
- Is the full text available?
- Is the article based on Cassandra?
- Is the article talking about metrics which signify the scalability?
- Is the article discussing performance of distributed databases?

Adding on to the literature review, sampling, both forward and backward is also done to find more relevant literature. Further, the same criteria have been used to scrutiny the literature obtained. After the final iteration of applying the criteria, a total of 40 research articles have been extracted. This literature review is conducted on the basis of [26] and is conducted in the following fashion.

1. Selecting the search strings: Searching multiple databases such as Inspec, Google Scholar and IEEE since the coverage of different databases is different in terms of literature and with keywords such as 'NOSQL' 'Cassandra' 'AutoScaling' 'AWS' 'Benchmark testing' 'Cassandra Evaluation 'cloud benchmarking' 'data scalability' 'Infrastructure as a service' are picked and searched for. Search strings such as (Latency) OR (CPU) AND Cassandra, Cassandra AND EC2, AWS OR Amazon Webservices AND (Cassandra), Performance AND Cassandra are formulated based on the topic selected to find more relevant literature and documentation. New keywords are obtained from the results yielded and are used to find more relevant literature.

2. Knowing and Analyzing the literature: The process of knowing the literature is to define and describe the available literature. Analysis of the yielded literature is done by separating, comparing and explaining the literature from the other less important pieces of literature.

3. Evaluating the Literature: Assessing or judging the literature by its relevance to this research is done in this step. Meaning that if the works of literature that are narrowed down are assessed against the present research and results to be explaining the concepts and the theories that can be used as a part of this research, then these pieces of literature are narrowed down.

4. Writing a review: The final step of the literature review is to extract the information yielded in a well-defined and structured piece of writing. This writing comprises of rational logic with thorough motivations and is organized to make the reader understand the context and the subject as well.

As a part of this literature review, Cassandra and Cassandra stress documentations of Datastax [28], [15], whitepapers on AWS [2] have also been reviewed extensively to understand the configurations of Cassandra clusters and to find different ways to deploy a Cassandra virtual nodes on AWS platform. Furthermore, for validation of the experiment, several iterations of the experiment have been made keeping in mind that AWS is a pay-as-you-go model.

3.2 Experimentation:

Experimentation is one of the commonly used research method for research. Planning and designing the experiments are done by the ideas given by literature review. Conventionally, experimentation is a research method where it investigates the considered variables that are affected during the experimental simulations [29]. In this research method of experimentation, the process is to assess a model or a system for its performance or specifically running simulations that would give insights on how the model or the system behaves or is affected by the variables considered. To put the aforementioned into this research context- since this study deals with performance of Cassandra scalability, the parameters or the metrics selected to evaluate the scalability are latency and CPU utilization. The Cassandra specific metrics namely replication factors and consistency levels are also considered since the research goal is to find these variables' affect on autoscaling of Cassandra instances.

Since the research method is a *controlled experimentation*, meaning that the practitioner has some control over the conditions where the study takes place, the researcher is in control to manipulate the independent variable [29]. The subject is treated analogously with regard to all variables except the independent variables.
Therefore, the *Independent variables* in this experiment are replication factors, consistency levels and types of various EC2 configurations used (t2.micro and t2.medium). The *Dependent variables* are therefore the CPU utilization and latency (ms).

Alternatives Rejection Criteria:

Survey: The survey results can be triangulated for market research and polls for a certain opinion but shall not play a role in selecting a model. Surveys are often conducted when the model or a tool is already in existence and the implementation is solely based on the choice of the practitioners [30]. In this study, since the scalability of Cassandra when different Cassandra specific parameters are considered cannot be defined by sample, this method is rejected.

Case Study: Case study is a research method used to investigate a single phenomenon is a particular time frame. Usually suitable for industrial evaluation of software models or tools to avoid the scale up problems. The difference between case study and experimentation is that the experiment is conducted to sample over the variables that being manipulated by the practitioner while on the other hand, a case study selects the variables representing the typical situation.

Hence, rejecting the above two research methods, experimentation with a mix of literature review is chosen as an apt research method for this research plan. The following flowchart explains the process of a *controlled experimentation*.

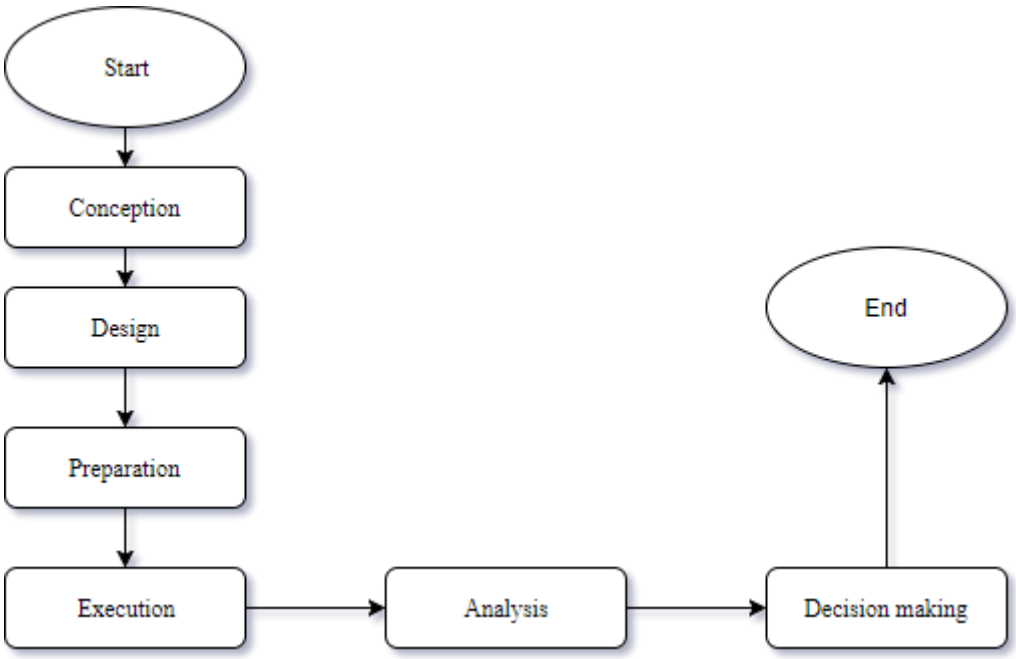

Figure 3-1: Flow of a controlled experiment.

This chapter describes the experimentation that has been carried out as a part of this research. The following sub-sections are strategically divided to enhance the understanding of the experimentation.

In this experiment, to answer the research question for determining the impact of replication factors, a maximum replication factor of 3 is selected for a three-node cluster and a maximum of 5 is selected for a five-node cluster. Therefore, only three and five node Cassandra clusters are autoscaled.

3.2.1 Experimental Setup

The experimental setup primarily consists of three EC2 instances on which Cassandra has been deployed. The whole set up is made to run on Ubuntu 14.04LTS operating system on AWS. The reason behind choosing Ubuntu 14.04 is that it is free-tier eligible. Then these three instances are configured to communicate and distribute the traffic to each other. The changes in the configuration is made in the YAML manifest which provides the settings of the parameters that can be changed. Thus, a three-node cluster is instantiated. Since there is no single point failure and no network bottlenecks, every node in the cluster is identical [31].

Primarily, one node out of the three available nodes is named master and the other two nodes are named worker nodes (W1 and W2). After logging into the instances, and accessing the YAML manifest, the following changes are made to ensure the communication between the seed and the two non-seed nodes.

On the Seed node:
-**Seeds**: IP address of the respective node
-**Listen_address**: IP address of the respective node
-**RPC_address**: IP address of the respective node.

On the other worker nodes (W1 and W2):
-**Seeds:** IP address of the seed node
-**Listen_address**: IP address of the respective node
-**RPC_address**: IP address of the respective node.

Then, the objective is to autoscale the three-node cluster over to five nodes. Therefore, traffic from Cassandra stress is directed to the master node. Then, a million writes are used to write and generate the keyspaces within the databases and 609 rate threads have been used in all experiments since the performance of Cassandra is noticed to be optimum. The traffic gets redistributed among the nodes present in the cluster in a round robin fashion and hence CPU utilization values on each node are recorded in a time series for the experimental duration of 15 minutes i.e. 900 seconds.

Further, determining the CPU utilization values for each of the Cassandra specific configurations that is for RF1CL1, RF2CL1, RF3CL1, RF3CLQ for a three-node cluster will allow the user to give the average CPU utilization as a threshold value to AWS for autoscaling. The representation of this setup is illustrated below.

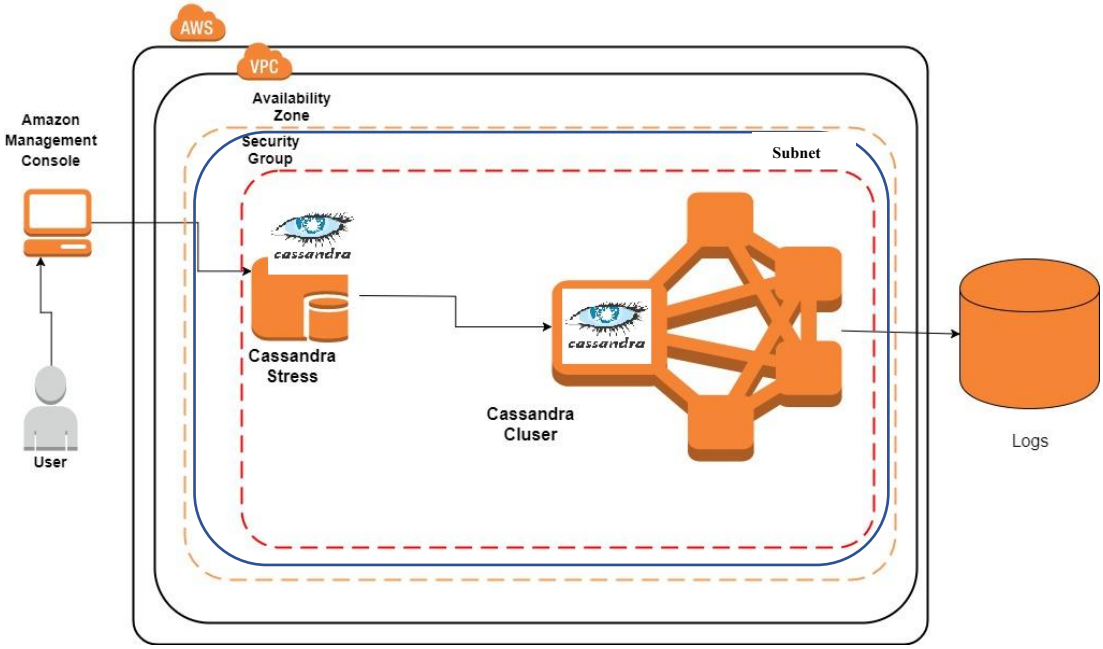

Figure 3-2. Experiment flow to find the threshold of each configuration

The same experimentation setup has been used to test the five-node cluster and scale it over to seven-nodes, as well. The Cassandra specific configurations that are taken into consideration in this thesis study are RF1CL1, RF3CL1, RF3CLQ, and RF5CLQ. During this phase of experimentation, it is required to define few configurations in the launch configurations on AWS. It is necessary that the nodes in the cluster bootstrap and gossip with the dynamically generated Cassandra nodes after autoscaling.

An additional Cassandra instance is instantiated with no relation to the existing cluster and the following changes have been made on the YAML manifest.

-Listen_address: newIP
-RPC_address: new IP

This node with newIP intentionally injected into the Listen and RPC address fields are created into an image and are given to the launch configuration to launch instances exactly like this one.

For the gossip to happen between the generated nodes and the existing cluster, a short snippet of bash script is inputted in the '**user data**' section in the launch configuration. The functionality of this bash script is primarily to retrieve the generated instance's metadata and to identify the YAML manifest on it. The Listen_address and RPC_address of these two generated nodes are set to be replaced from newIP to the instance's own private IP and to run Cassandra on the generated nodes. This ensures the communication between the generated nodes and joins the cluster. The same experiment is repeated for each Cassandra specific configuration iteratively to ensure that the results are valid.

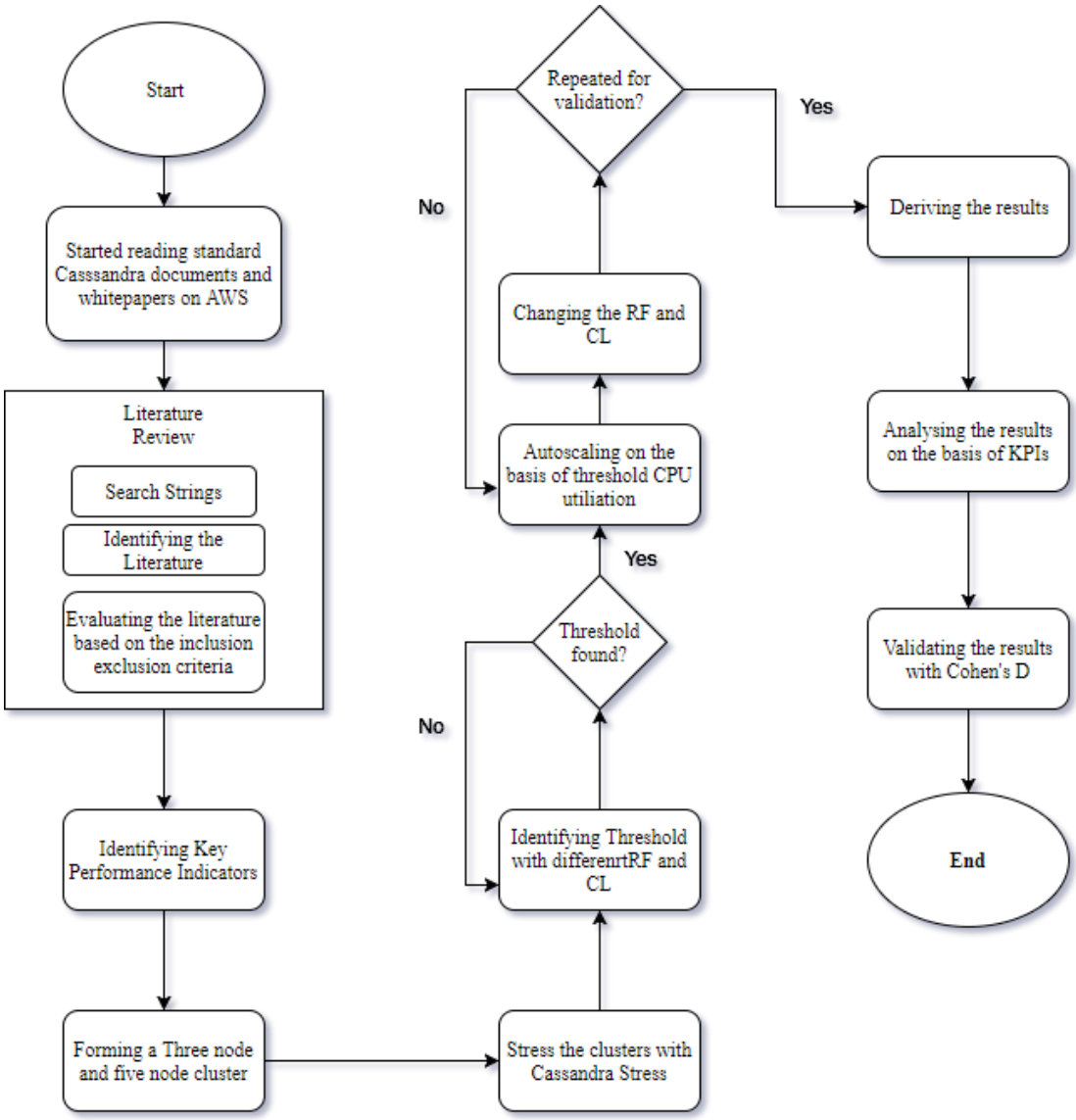

Figure 3-3: The overall experimentation flow

3.2.2 Designing the Experimentation

The following points shall explain elaboratively as to how load test is carried out, how replication factors and consistency levels are tweaked in the experimentation.

- Primarily from the Cassandra stress instance, records are created on the cluster under test. This can be done by passing a scenario from the stress node as follows.

  ./cassandra-stress write n=1000000 -node $IPaddress

- This is followed by running a mixed workload of equal reads and writes for the duration of 15 minutes. The following command executes a mixed workload and saves the log file of the load run on a .csv file.

  ./cassandra-stress mixed ratio\(write=5,read=5\) duration=15m cl=ONE -pop dist=UNIFORM\(1..1000000\) -log file=~/logs5/3dummy.csv -node $IPaddress -rate threads=609

- On each of the nodes in the cluster, the following dstat command is executed to monitor the instances for CPU utilization along the timestamp.

  dstat -c -t --output nodeX.csv

- In the meantime, when the CPU utilization of the cluster reaches the threshold value for 300 seconds consecutively [7], then the EC2 instances scale over to 5 nodes and the same dstat command as above is made to run on the newly generated nodes to monitor the behavior of these new instances.

- Further, to change the replication factor, a new keyspace is created on the seed node and records are newly created based on this new keyspace. This can be done by executing the following command from the stress node.

  ./cassandra-stress write n=1000000 -schema keyspace="NewKeyspace" -node $IpAddress

- To change the consistency levels to TWO, QUORUM or ALL and to run a mixed workload, the following command is executed from the stress node.

  ./cassandra-stress mixed ratio\(write=5,read=5\) duration=15m cl=QUORUM -pop dist=UNIFORM\(1..1000000\) -log file=~/logs5/Quorum.csv -node $IPaddress -rate threads=609

- The same experiment is iteratively repeated for each of the configurations and the logs are saved for analysis in terms of latency and CPU utilization and to evaluate the scalability.

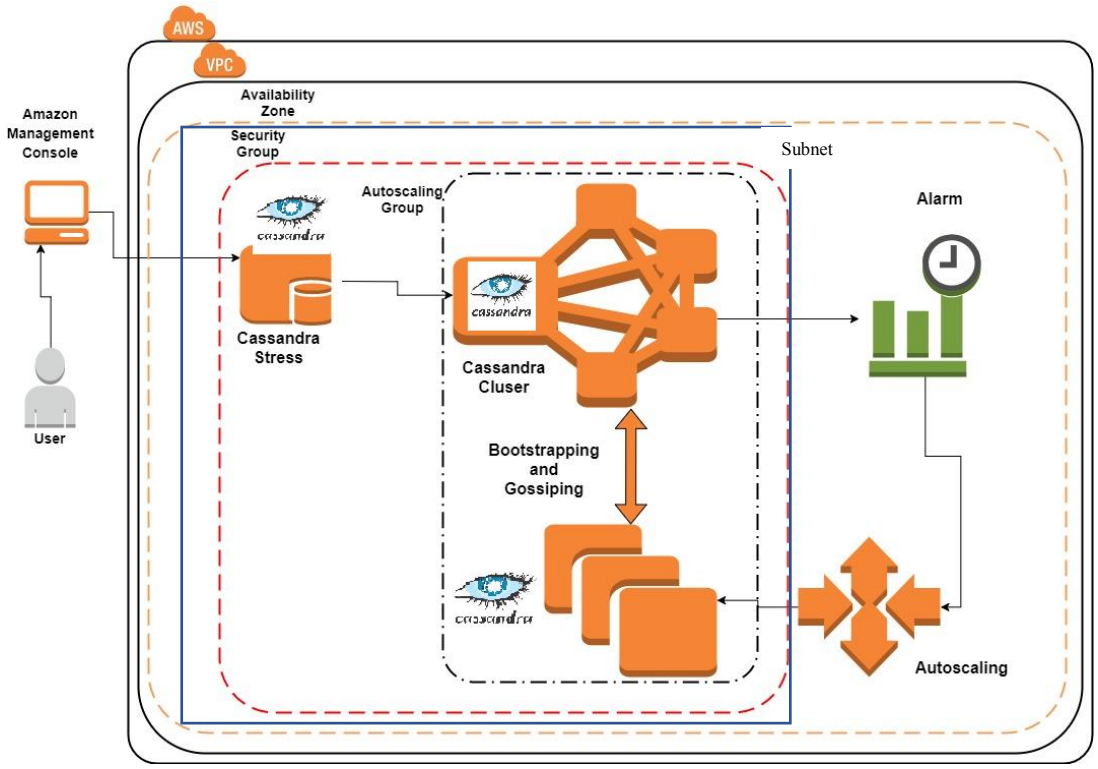

Figure 3-4. Experiment flow to autoscale each configuration

- The figure above describes the flow of the experimentation and gives insights of the primary outcome of this experiment which is to scale the cluster over without any downtime.

4 RESULTS

The following are the results of the experimentation that has been carried out as a part of this research. The first part of the results shows the mean CPU utilization values for the experiment that has been repeated iteratively for 5 times, in order to find the threshold value. These threshold values are then used to autoscale the 3 and 5 node clusters respectively.

4.1 Threshold CPU utilization values for a 3-node cluster:

The experiment for a three-node cluster with the changing of RF and CL has been done iteratively for 5 times each and the following are the values of the mean CPU utilization values on t2. medium instances and the respective standard deviation values. The appropriate statistical analysis has been carried out in order to determine the threshold value to breach. The following table shows the mean CPU values for different configurations and their respective standard deviation values for a 3-node cluster.

Configuration	Mean CPU Utilization	Standard Deviation
RF1CL1	79%	2.70
RF2CL1	89%	4.82
RF3CL1	92%	2.88
RF3CLQ	89%	3.97

Table 4-1 Mean CPU utilization and standard deviation values **before** autoscaling for a **3 node cluster**

4.2 Threshold CPU utilization values for a 5-node cluster:

On the other hand, a 5 node cluster has the following mean CPU utilization values for different configuration types in a 5 node cluster. Also, their respective standard deviation values have also been listed in the table.

Configuration	Mean CPU Utilization	Standard Deviation
RF1CL1	56%	2.91
RF3CL1	93%	3.21
RF3CLQ	92%	2.77
RF5CLQ	89%	3.03

Table 4-2 Mean CPU utilization and standard deviation values **before** autoscaling for a **5 node cluster**

The following depicts cumulative distribution curves that show the effect of autoscaling on the system. Different plots for different Cassandra specific configurations have been plotted and analysis has been carried out triangulating the results of all iterations of the experiment. All the experiments have the same duration of 900 seconds and have been subjected to the same workload of equal reads and writes and with the same number of threads i.e. 609.

4.3 Experimental results for a three-node Cassandra cluster:

The following graphs are plotted for a three-node cluster when scaled over to five nodes.

4.3.1 For the configuration Replication Factor1 and Consistency Level1:

Below is the cumulative distribution curve for latency for three nodes that is before scaling and five nodes that represent after scaling. The X-axis represents the time in seconds and the Y-axis represents total mean latency in milliseconds.

The threshold value to breach in this case is identified as 79%. Therefore, when the average CPU utilization reaches and remains at 79% for 300 consecutive seconds, then two newly generated nodes are added to the existing cluster. The 3-node cluster with RF1 and CL1 configurations has autoscaled at 629th second which means that the second line in the graph after the 629th seconds represent the total latency of five nodes including the two newly generated ones.

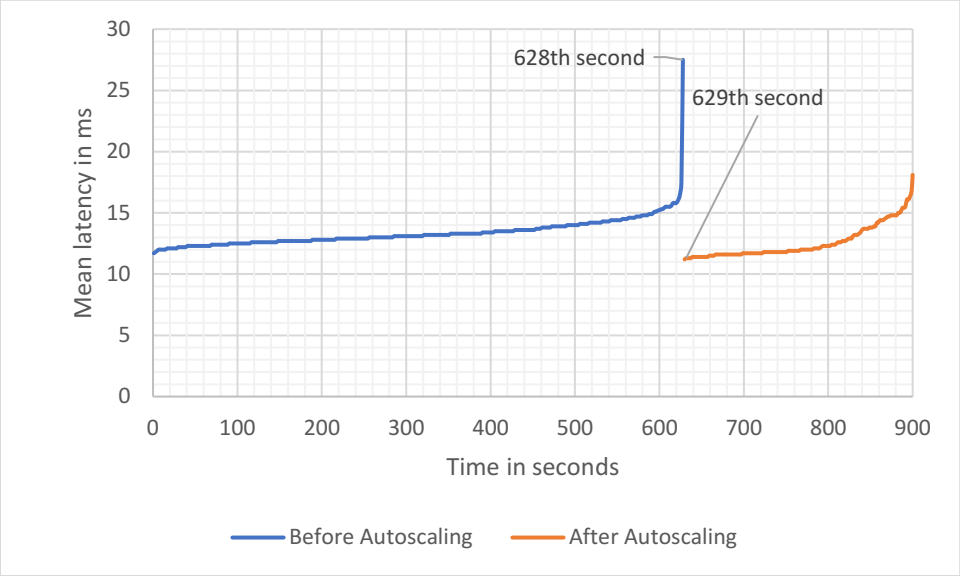

Figure 4-1. Total Latency of active nodes for RF1 and CL1

From the cumulative CPU utilization graph below, it can be visualized that the threshold value is 79% and it been scaled over to five nodes when the mean utilization is maintained at 79%. The line which represents the behavior before autoscaling is the representation of values of CPU utilization averaged over three nodes and the line representing the behaviour after autoscaling is the representation of the CPU utilization values averaged over five nodes.

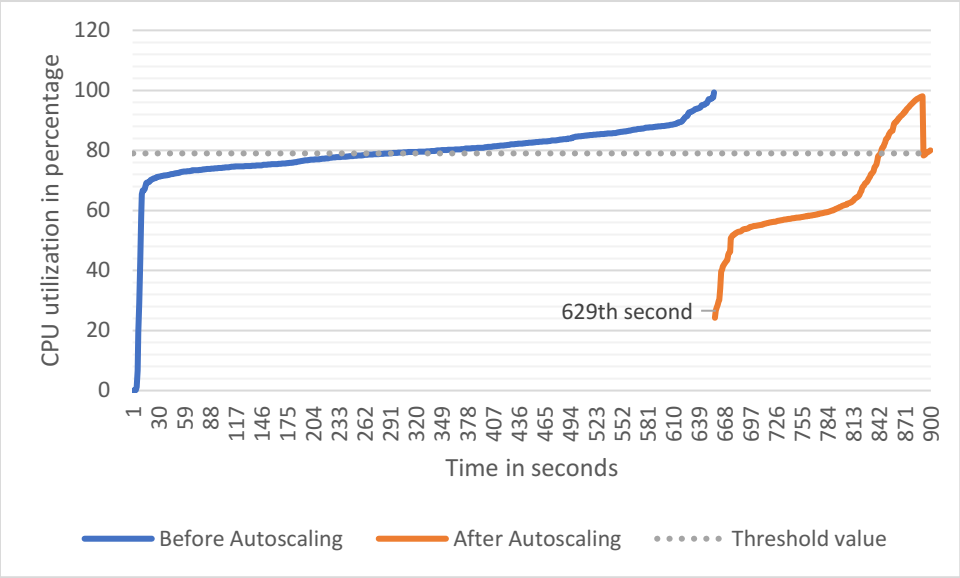

Figure 4-2. Averaged CPU Utilization of active nodes for RF1 and CL1 configuration

4.3.2 For the configuration Replication Factor3 and Consistency Level Quorum:

For this configuration, the threshold value to breach is 86% and the cluster has autoscaled at 461st second after the experimentation has been triggered. The total latency has been recorded in the following figure which represents time in seconds on X-axis and Y-axis represent the total mean latency in milliseconds. The labels in the graph area represent the time of autoscaling.

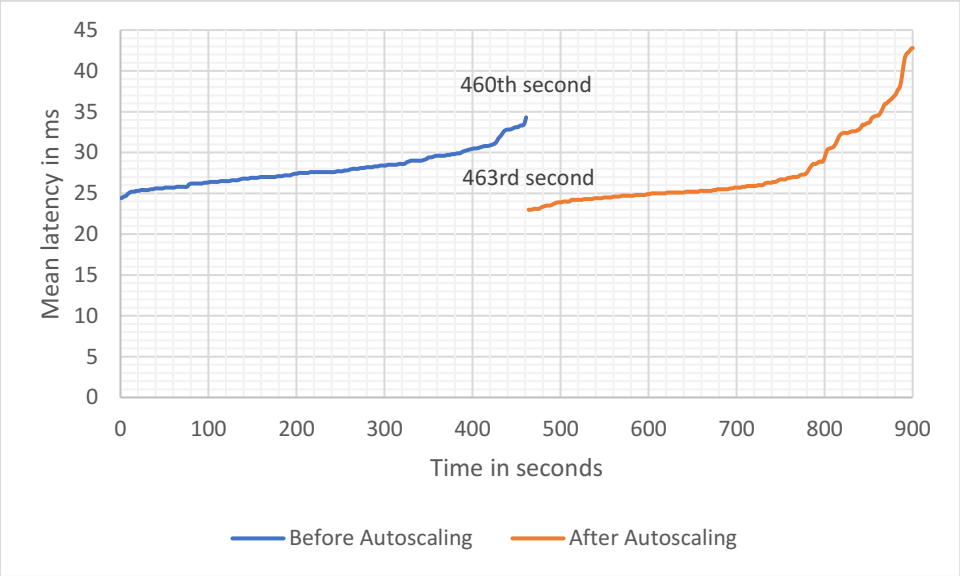

Figure 4-3. Total latency of active nodes for the configuration RF3CLQ

For the CPU utilization curves, the plots on X-axis is the time in seconds and the plots on Y-axis represents the mean CPU utilization of the active nodes in the cluster before and after autoscaling. The following figure shows the behavior before the autoscaling, after the

autoscaling and the threshold value to breach as well. The time of autoscaling is 463rd second after the experimentation has been triggered.

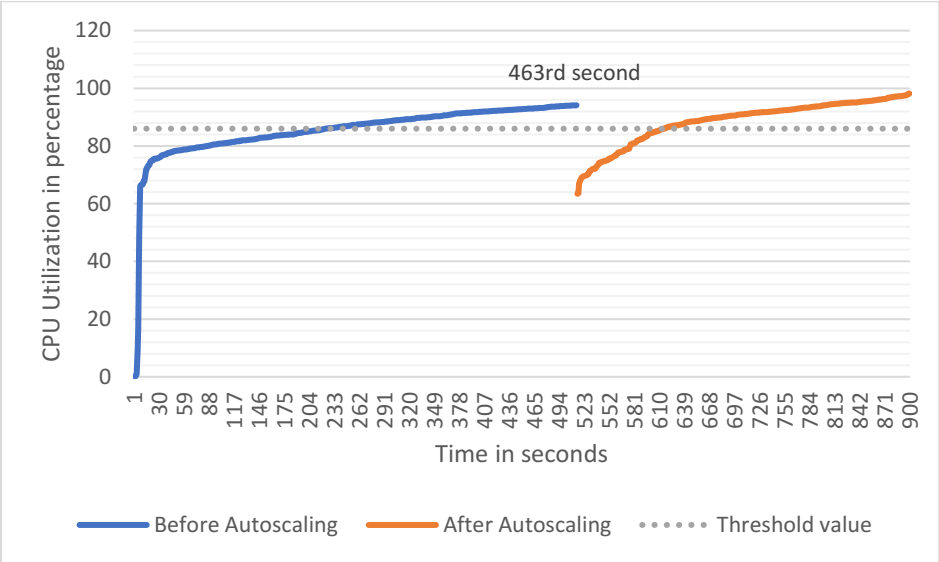

Figure 4-4. Averaged CPU Utilization of active nodes for RF3 and CLQ configuration

4.3.3 For the configuration Replication Factor 3 and Consistency Level 1:

The threshold value in this case, has been identified as 91% and is maintained as the mean utilization value to autoscale. The following figure represents the total latency before autoscaling and the total latency after autoscaling on a cumulative distribution graph. The time of autoscaling is at the 606th second after the experimentation has been triggered.

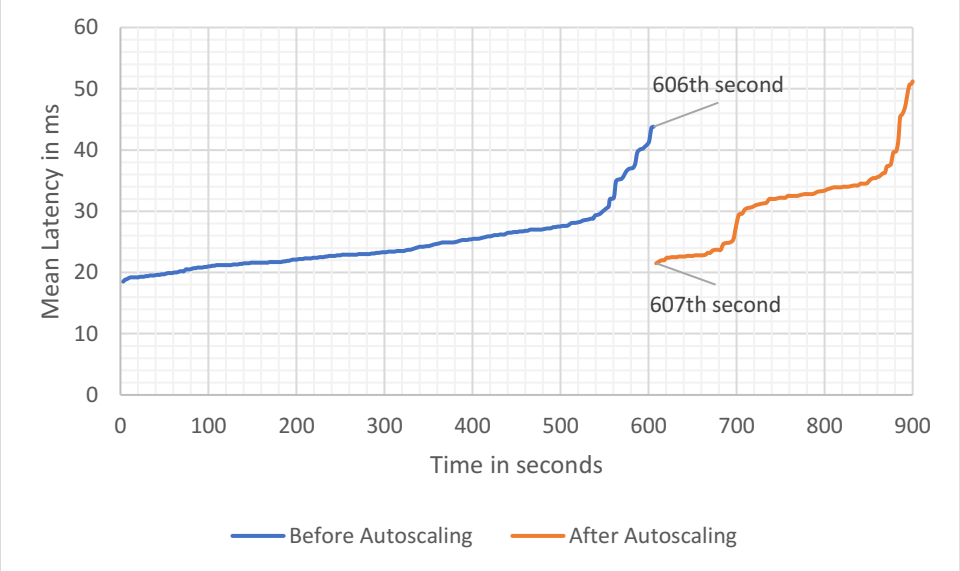

Figure 4-5. Total Latency of active nodes for the configuration RF3 and CL1

The below cumulative distribution graph represents the CPU utilization of the system before and after autoscaling. The threshold value to breach which is 91% has also been shown on this graph.

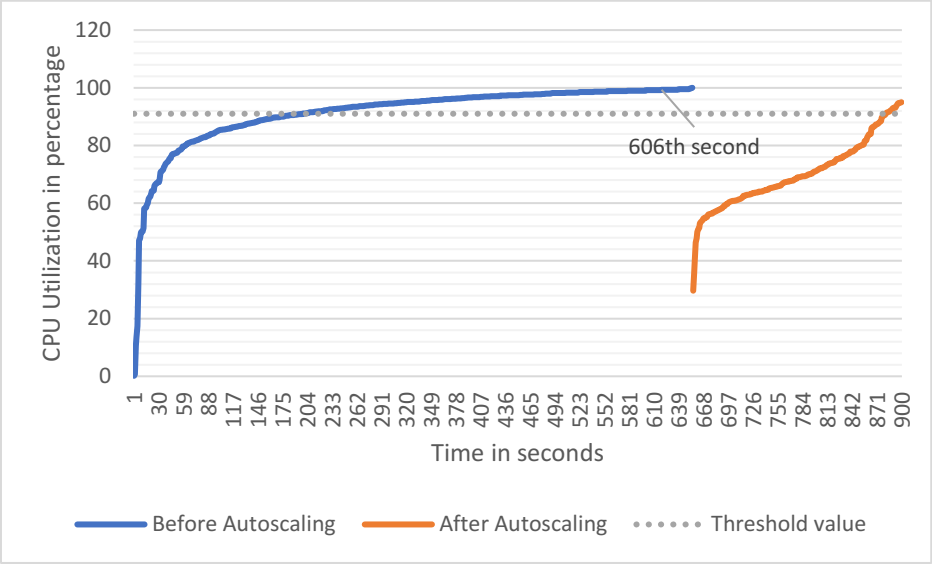

Figure 4-6. Averaged CPU Utilization of active nodes for RF3 and CL1 configuration

4.3.4 For the configuration Replication Factor 2 and Consistency Level 1:

The threshold value to breach is 89% mean CPU utilization and the cluster has autoscaled at the 416th second after the experimentation has been triggered. The X-axis represents the time in seconds and the Y-axis represents the mean latency in milliseconds

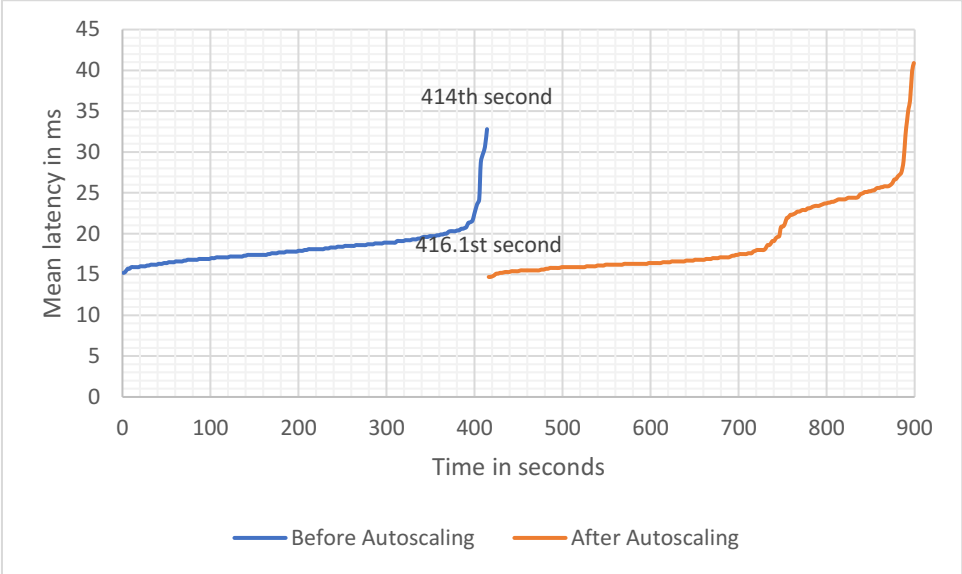

Figure 4-7. Total Latency of active nodes for the configuration RF2 and CL1

The following CPU utilization graph illustrates the cluster before autoscaling and after autoscaling. The threshold axis has also been illustrated which is at 89%.

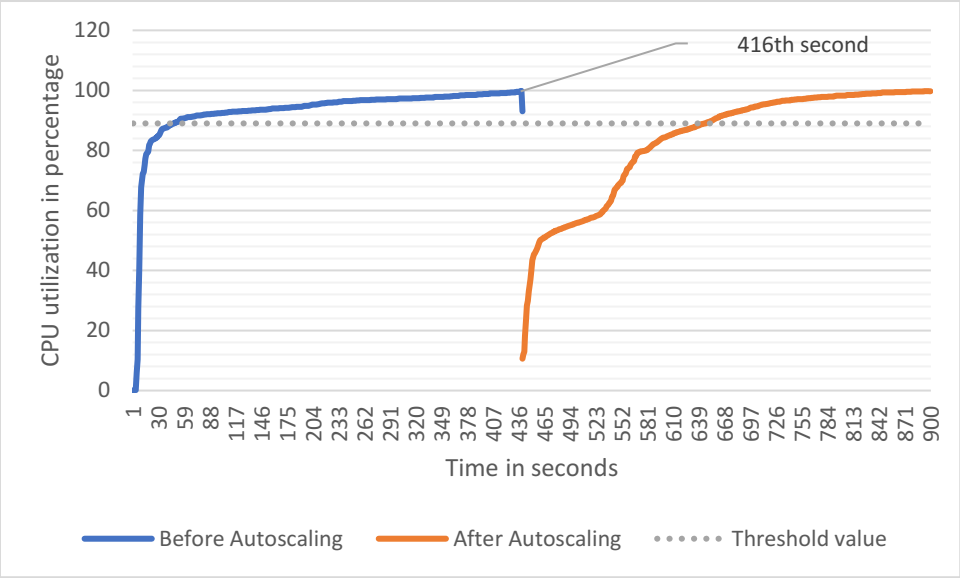

Figure 4-8. Averaged CPU Utilization of active nodes for RF2 and CL1 configuration

The following section of graphs is plotted for a five-node cluster when scaled over to seven nodes.

4.4 Experimental results for a five node Cassandra cluster

The following graphs are plotted for a five-node cluster when scaled over to seven nodes.

4.4.1 For Replication factor 1 and Consistency Level 1:

For a five-node cluster with the configuration Replication Factor 1 and Consistency Level 1, the threshold was identified as 58% of CPU Utilization and therefore this five-node Cassandra cluster will autoscale to seven nodes when the mean CPU Utilization on all the five nodes reach 56%. Below is the total latency graph where it shows that the cluster has autoscaled after the 338th second from the experiment has started. The X and Y axes represent the mean latency in milliseconds and time in seconds respectively.

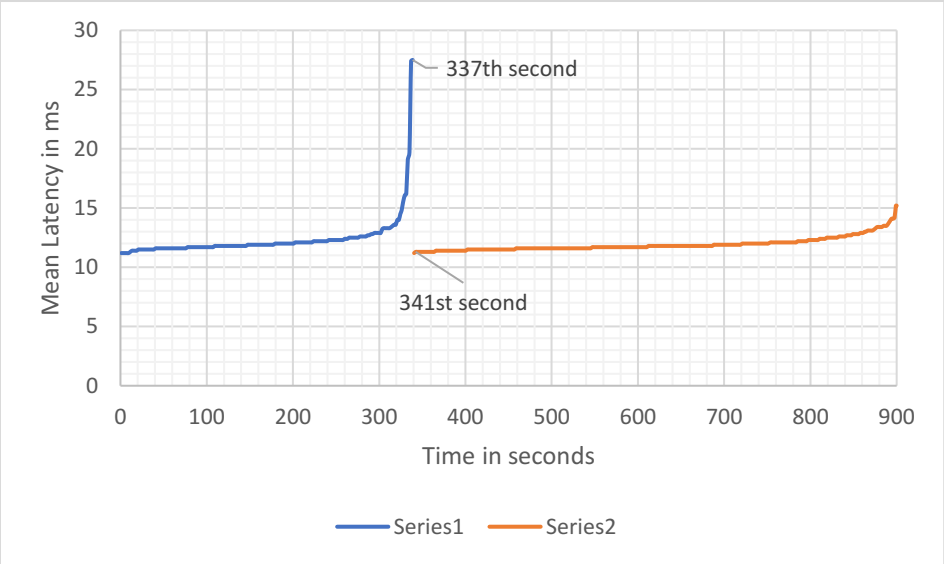

Figure 4-9. Total latency of active nodes for the configuration RF1 and CL1

The following CPU Utilization graph illustrates. This graph illustrates the case before and after autoscaling. Another axis, used as a reference line describes the threshold at 56%. However, since the number of nodes after autoscaling is seven, the time taken for Cassandra to bootstrap to the existing cluster took a considerable amount of time. This bootstrapping time is calculated to be 70 seconds and only after that period of 70 seconds, the seven-node Cassandra cluster started to gossip and communicate.

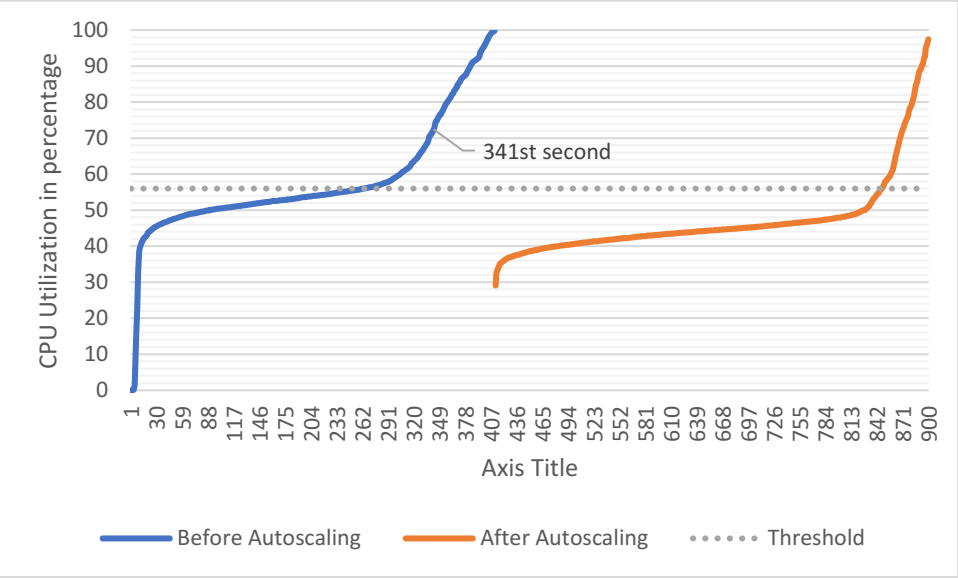

Figure 4-10. Averaged CPU Utilization of active nodes for RF1 and CL1 configuration

4.4.2 For Replication Factor 3 and Consistency Level 1:

For Replication Factor 3 and Consistency Level 1, the threshold to breach the alarm was identified to be 93% of CPU Utilization, the cluster autoscaled over from five nodes to seven nodes after 93% utilization was reached at 431st second from the experimentation has been started. The following total latency graph illustrates the result for the configuration RF3 and CL1.

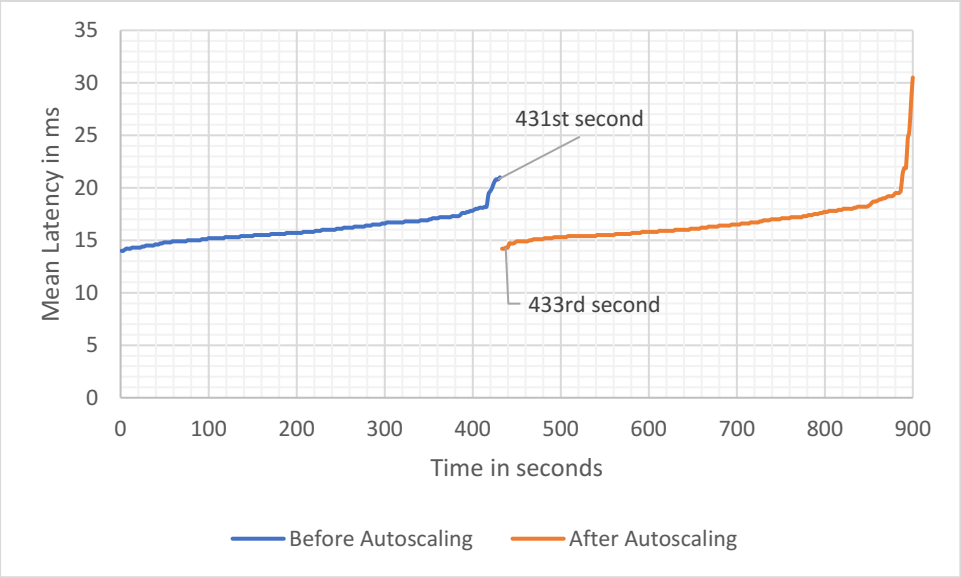

Figure 4-11. Total latency of active nodes for the configuration RF3 and CL1

Below is the graph which depicts the CPU Utilization for before autoscaling, threshold and after autoscaling. However, the cluster has autoscaled at 431st second. The time taken for bootstrapping the newly generated Cassandra nodes to existing cluster and start communicating was about 90 seconds for this configuration. The X and Y axes represent CPU Utilization and time in seconds respectively.

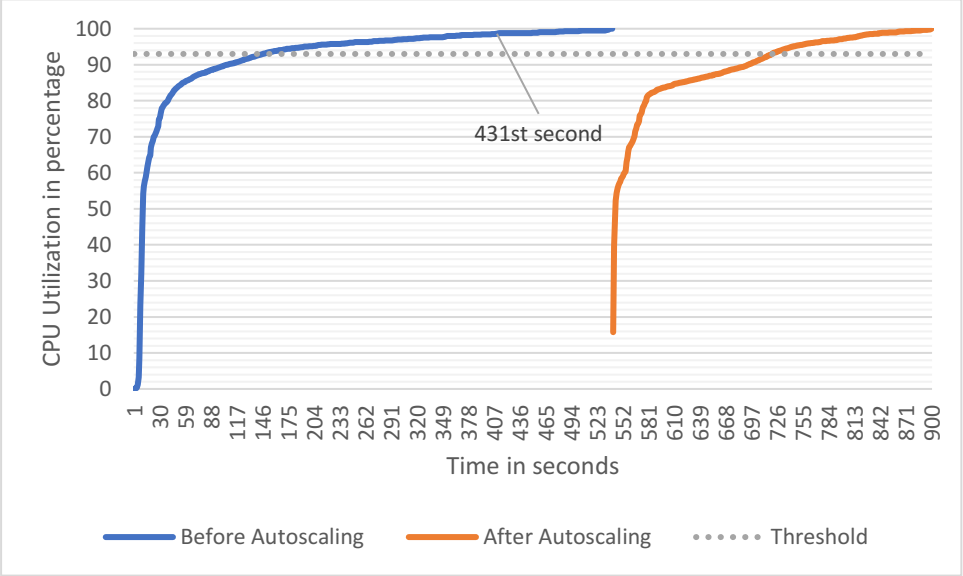

Figure 4-12. Averaged CPU Utilization of active nodes for RF3 and CL1 configuration

4.4.3 For Replication Factor 3 and Consistency Level Quorum:

The threshold to breach in this case was 92% of CPU Utilization and this utilization was reached after 391 seconds of experimental duration, the Cassandra cluster autoscaled over to seven nodes. The following graph illustrates the total latency curves before and after autoscaling and it can also be implied from the graph that the cluster has scaled after the 391st second.

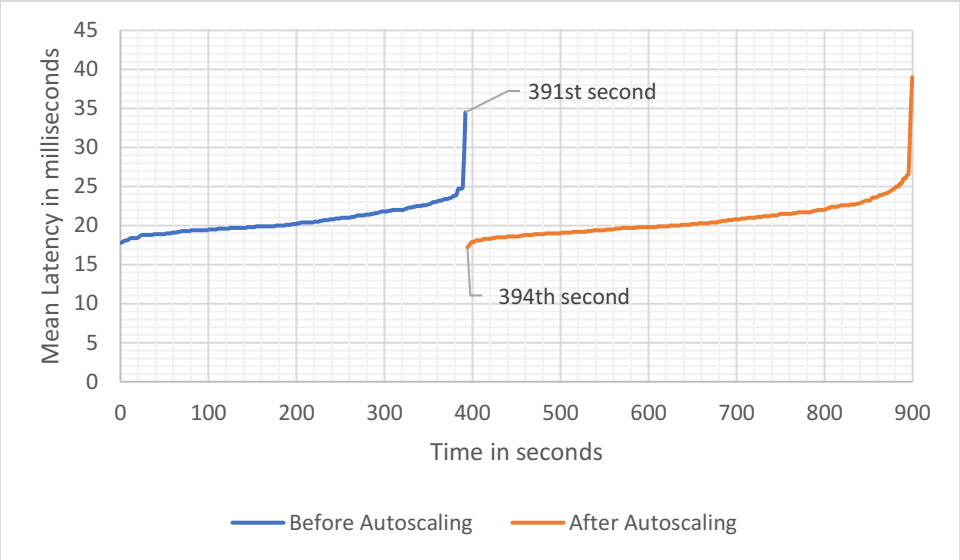

Figure 4-13. Total latency of active nodes for the configuration RF3 and CLQ

The below CPU utilization graphs indicate the utilization before and after autoscaling and the threshold at which the alarm breached. In this configuration, the newly generated nodes after autoscaling took about 100 seconds to bootstrap to the existing Cassandra cluster. This bootstrapping time is relatively higher than the configurations above due to the consistency level being quorum.

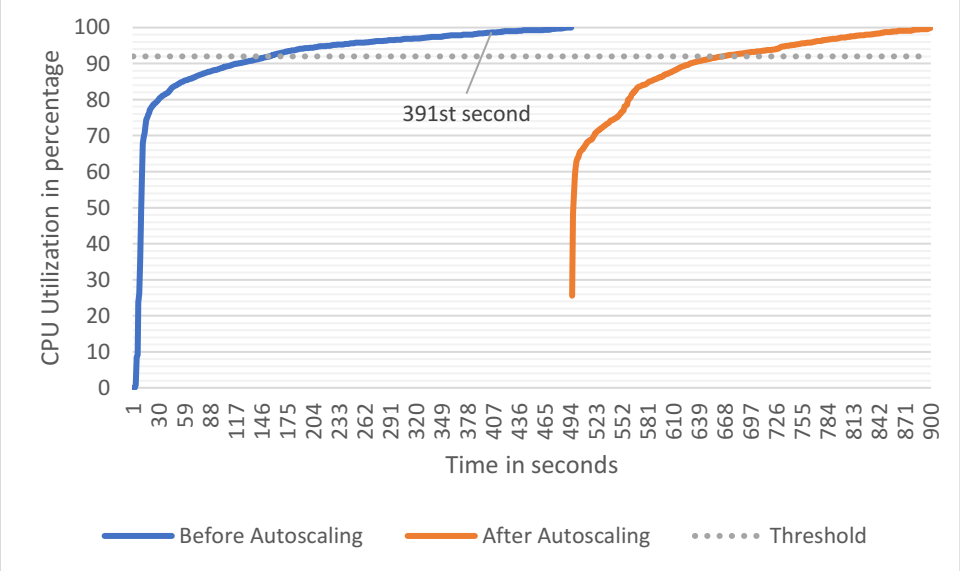

Figure 4-14. Averaged CPU Utilization of active nodes for RF3 and CLQ configuration

4.4.4 For Replication Factor 5 and Consistency Level Quorum:

For the Replication Factor 5 and Consistency Level of Quorum, the following total latency graph indicates that 5-node Cassandra cluster has autoscaled after 330 seconds from the experimentation has been started. The two lines in the graph describe the cases before and after autoscaling respectively. X and Y axes represent latency in milliseconds and time in seconds respectively. The threshold at which the alarm breached and the cluster autoscaled was at 89% of CPU Utilization.

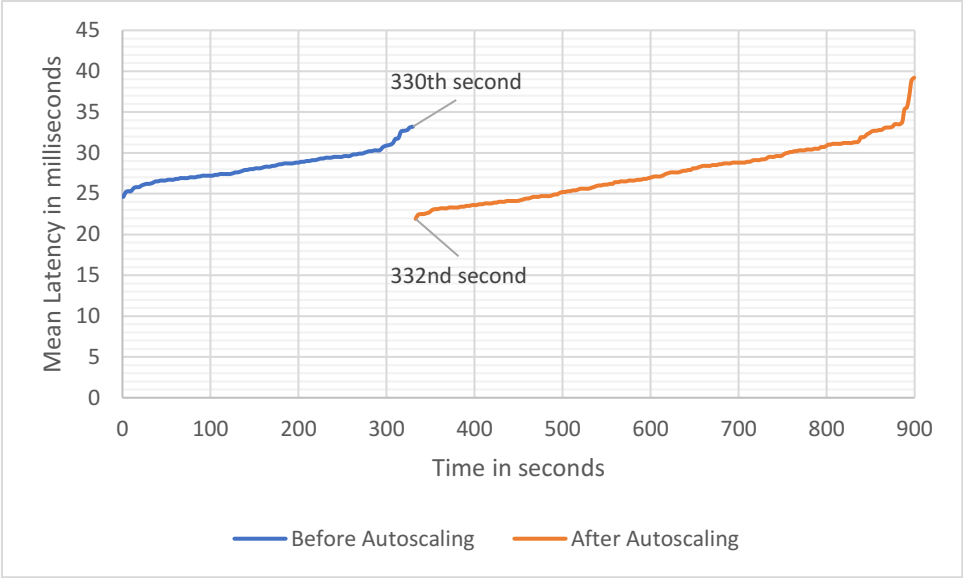

Figure 4-15. Total latency of active nodes for the configuration RF5 and CLQ

The below CPU Utilization graph indicate that the newly generated nodes after autoscaling took about 78seconds to bootstrap to the existing five-node cluster. The threshold and the cases before and after autoscaling have been illustrated in the graph below.

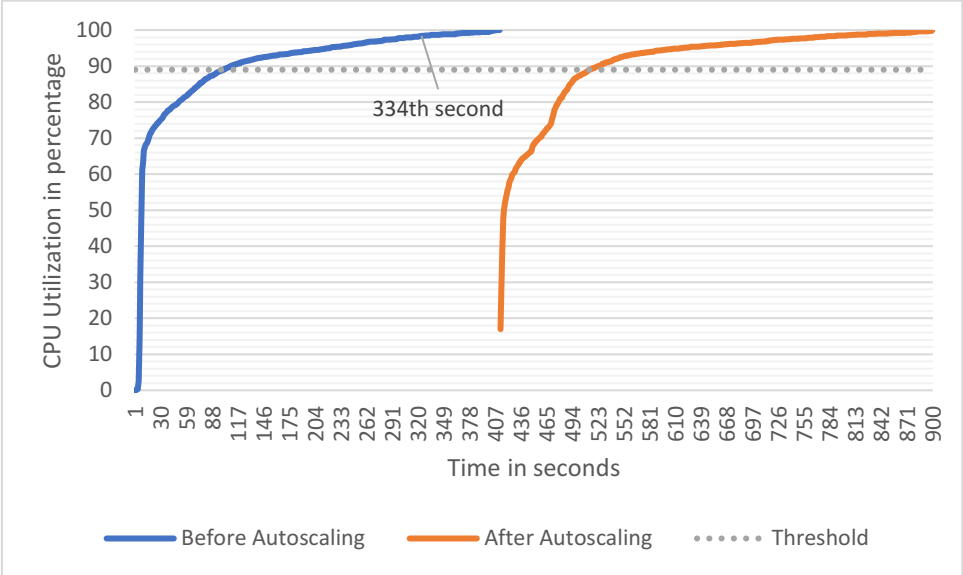

Figure 4-16. Averaged CPU Utilization of active nodes for RF5 and CLQ configuration

The mean latencies and the CPU utilizations categorized for different configurations and for before and after autoscaling with their respective Cohen's D values are mentioned below in the Analysis and Discussion chapter.

5 ANALYSIS AND DISCUSSION

This chapter describes the analysis of the results yielded and reasons with them. The first subsection of this chapter is the analysis wherein results are analyzed on the basis of Latency and CPU Utilization and the second subsection is discussions wherein answers to the research questions are formulated. Validity threats follow the discussions.

5.1 Analysis

This chapter explains the process of analysis of the results yielded. The results are categorized based on selected metrics, Latency and CPU utilization to show the effects of Autoscaling.

5.1.1 Latency

It has been observed that there is a decrease in the mean latency in all the configurations of Cassandra. This decrease in latency can be directly related to the elasticity of Cassandra as it has been scaled over five nodes. The mean latency gain for different configurations in a 3-node and 5-node Cassandra cluster can be inferred from the table below. It can be also be seen that the latency difference attributed to autoscaling is evident.

.

In a three-node Cassandra cluster scaling over to five-nodes, for RF2CL1, the latency difference is higher when compared to the other configurations. This is because in a configuration of replication factor being two and consistency level being one, each of the node holds 67% of the data before autoscaling and holds 40% of data after the cluster autoscales.

For RF3CLQ configuration, the latency is the highest among all the configuration. The reason behind this is that each node holds 100% of the data before autoscaling and becomes 60% after the cluster autoscales. But since the consistency level is maintained at QUORUM, writing and reading the data is done from two nodes. Hence the time for each memory request is higher than the rest all configurations. Below is the table where the mean latencies for different Cassandra specific configurations are mentioned.

For a 3-node Cassandra Cluster scaling over to five nodes		
Configuration Type	**Before Autoscaling**	**After Autoscaling**
RF1CL1	13.35ms	13.30ms
RF2CL1	18.28ms	16.9ms
RF3CL1	24.43ms	23.72ms
RF3CLQ	27.96ms	27.40ms

Table 5-1. Mean latency in milliseconds for each configuration before and after autoscaling in a three-node cluster.

In a five-node Cassandra cluster scaling over seven-nodes, for RF3CL1, the difference in latency before and after autoscaling is higher in the five-node cluster category. The latency gain is about 0.67ms and is almost twice as of RF5CLQ, and RF1CL1. This difference can be related to the difference in replication of data. Before autoscaling, each node in the cluster holds 60% and the after autoscaling each node holds 43% data. The difference is storage itself is 17%. Therefore, the request to response time requires less time.

The latency difference between the before and after autoscaling for both RF5CLQ, and RF1CL1 is about 36ms and is relatively higher than the default configuration of RF1CL1. This difference for RF5CLQ can be attributed to the replication factor being five (5) which means that the data is replicated in all the 5 nodes before autoscaling. This means that each node before autoscaling holds 100% of the data and after autoscaling, due to the replication factor being 5 even when the cluster has scaled over to 7 nodes, each node holds about 71% of the data.

This difference in replicating and querying data can be attributed to the latency difference. For the configuration RF1CL1, each node before autoscaling holds 20% of the data and after autoscaling, each node holds 14% of the data. This difference in replicating can be attributed to the difference in latency before and after autoscaling. For the configuration RF3CLQ, the latency gain is small but noticeable. This small change is because of the small replication difference and the nodes in this case, are consistent which means that all the nodes will be able to 'see' the same data.

For a 5-node Cassandra Cluster scaling over to seven nodes		
Configuration Type	**Before Autoscaling**	**After Autoscaling**
RF1CL1	12.30ms	11.94ms
RF3CL1	16.72ms	16.05ms
RF3CLQ	20.60ms	20.49ms
RF5CLQ	28.37ms	27.73ms

Table 5-2: Mean latency in milliseconds for each configuration before and after autoscaling in a five-node cluster.

5.1.2 CPU Utilization

This experiment has been repeated for 5 times iteratively which means that after taking the threshold value, the 3-node and 5-node clusters have been autoscaled 5 times for each configuration considered respectively. The following table illustrates the mean CPU utilization values after autoscaling and their respective standard deviation values for a 3-node cluster with varying the RF and CL configurations.

Configuration	**Mean CPU Utilization**	**Standard Deviation**
RF1CL1	72%	1.94
RF2CL1	81%	3.97
RF3CL1	83%	3.82
RF3CLQ	78%	1.30

Table 5-3 Mean CPU utilization and standard deviation values **after** autoscaling for **3-node cluster**

The following table shows the mean CPU utilization and standard deviation values for a 5 node cluster after autoscaling with manipulating different RF and CL.

Configuration	Mean CPU Utilization	Standard Deviation
RF1CL1	47%	3.53
RF3CL1	90%	2.12
RF3CLQ	85%	1.02
RF5CLQ	87%	2.94

Table 5-4 Mean CPU utilization and standard deviation values **after** autoscaling for **5 node cluster**

Quantifying the CPU utilization for each configuration for both before and after autoscaling cases gives an insight as to how much CPU has been decreased after autoscaling. The second research question to put forth the impact of RF and CL and to quantify their impact on the CPU utilization, the metric to be considered in this study, is done by deriving the Cohen's d. The decrease in the CPU utilization is also evident because autoscaling is adding additional CPUs and theoretically there can be an observed decrease in the mean utilization. To quantify the effect of autoscaling, analysis on the based-on Cohen's d values has been done. Cohen's D value is given by the formula,

$$\text{Cohen's } d = (M_2 - M_1) / SD_{\text{pooled}} \quad \text{where} \quad SD_{\text{pooled}} = \sqrt{((SD_1^2 + SD_2^2) / 2)} \text{ [32].}$$

**M1 and M2 are the mean values in both groups
**SD1and SD2 are the standard deviation values in both groups.

In a three-node Cassandra cluster scaling over to five nodes, the Cohen's d value for RF1CL1 configuration has been identified as 2.90 which indicates that the effect of autoscaling has high significance in this case. This can be directly related to the phase before autoscaling where in the mean of the three nodes is 79% and the mean after autoscaling is 72% and therefore the reduction in CPU utilization is 7%. The Cohen's d value for RF2CL1 configuration is only 1.81, lesser when compared to the RF1CL1 configuration.

The reason behind this is the mean CPU utilization over 3 nodes was 89% while the CPU utilization for 5 nodes was 81%. Therefore, with the replication factor being two and the decrease in utilization being 8% is the reason behind this small effect indicated by Cohen's d. For the configurations of RF3CL1 and RF3CLQ are relatively higher with the values 2.68 and 3.72 respectively with the difference in CPU utilization as 9% and 8% respectively. This means that there is a considerable change in the CPU utilization before and after autoscaling in the configurations of RF3CL1 and RF3CLQ configurations.

Configuration Type	Cohen's D value
RF1CL1	2.90
RF2CL1	1.81
RF3CL1	2.68
RF3CLQ	3.72

Table 5-5: Cohen's D value for each configuration in a three-node cluster.

In a five-node Cassandra cluster scaling over to seven nodes, the Cohen's d value for RF1CL1 configuration is 2.78 which suggests that the effect of autoscaling is significant. This can be associated with the mean CPU Utilization before autoscaling which was 56% and after the cluster autoscaled the mean CPU Utilization was found to be 47%. The Cohen's d value for the configuration RF3CL1 is 1.10 which suggests that there is very less or no effect of autoscaling. This is because of the mean CPU Utilization is already at 93% and after autoscaling the mean CPU Utilization has reduced by 3%. Therefore, the change is insignificant. For the configuration RF3CLQ, the Cohen's d value is 3.35 which means that

autoscaling has significant effect. Lastly, for the configuration RF5CLQ, the Cohen's d value is 2.34 for the same reason as above. The autoscaling has very less effect on this configuration as the CPU Utilization was 89% before autoscaling and with the data replicating in 5 nodes and the consistency level being Quorum, the CPU Utilization after autoscaling was 87%.

Configuration Type	Cohen's D value
RF1CL1	2.78
RF3CL1	1.10
RF3CLQ	3.35
RF5CLQ	2.34

Table 5-6: Cohen's D value for each configuration in a five-node cluster.

5.2 Discussions

This section compares and describes the conclusions that have been obtained in this thesis study with the previous research in this fields of performance evaluations. In this research performance of Cassandra scalability was evaluated on Amazon EC2 infrastructure. In[19], Gandini et al. have deployed Cassandra instances on Amazon EC2 and have compared the performances with Hbase and MongoDB. It has been found using an equal mixed ratio of reads and writes (50-50), the latency and throughput of Cassandra outperform in an optimum way when compared to HBase and MongoDB. It has also been mentioned that replication factors have been experimented with in Cassandra and MongoDB. Gandini et al. have shown that with the increase in replication factor, the latency and the throughput also increases. In the current study however, the effects of autoscaling on replication factors and consistency levels are shown and the results agree with that of [19].

In [33] Enqing Tang and Yushun Fan have compared the performance of Cassandra with Redis, MongoDB, Couchbase, and MongoDB. By using YCSB tool to stress the different NoSQL databases, they have concluded that horizontal scalability is the optimum choice to make when the data is growing rapidly. The mean CPU utilization values before and after autoscaling, in the current research, show the optimum usage of the EC2 instances. Therefore, it is indeed easy for horizontal scalability when the data is increasing rapidly.

In [18] Veronika Abramova et al. evaluate the performance of Cassandra in a parallel cluster environment. It has been discussed that Cassandra's latency keeps stable and does not differ in different cluster environments. In this study, the Cassandra specific configuration of Replication Factor 3 and Consistency Level 1 has been considered for two different clusters while autoscaling. The two different configurations show two different latency times. This shows that there is an impact of autoscaling on the replication factors and consistency levels.
In [34], Jackson et al. perform analysis of high-performance applications on Amazon EC2 infrastructure. The research shows that there is a variation in Amazon EC2 itself and it is significant. The variation of the performance is due to the shared nature of the virtualization by the network. This concern is considered in the current research and the experiment has been repeated iteratively to avoid the performance variation as much as possible.

5.2.1 Answers to research questions

In this chapter discussions along the lines of research questions has been given and further, threats to validity with respect to this experimentation has been addressed. The following are the research questions that are a part of this research.

RQ1) How to measure and what is the latency in adding Cassandra nodes to Amazon EC2 instances?

Autoscaling the Cassandra nodes from three nodes to over five nodes and measuring the latency with the Cassandra stress tool before and after autoscaling will allow to quantify the latency in adding Cassandra nodes to the Amazon EC2 instances. This experimentation has been repeated iteratively for different configurations of Cassandra. The mean latency has been quantified based on the Cassandra specific configurations as follows.

- For RF1CL1, the mean latency after autoscaling is identified as 13.30ms.
- For RF2CL1, the mean latency after autoscaling is diagnosed as 16.9ms.
- For RF3CL1, the mean latency after autoscaling is found to be 23.72ms.
- For RF3CLQ, the mean latency after autoscaling is determined as 27.40ms.

Furthermore, the Cassandra nodes from five nodes have been scaled over to seven nodes. The results of the mean latencies for different Cassandra specific configurations have been identified as follows.

- For RF1CL1, the mean latency after autoscaling is identified as 11.94ms.
- For RF3CL1, the mean latency after autoscaling is 16.05ms
- For RF3CLQ, the mean latency after autoscaling is found to be 20.79ms
- For RF5CLQ, the mean latency after autoscaling is determined as 27.72ms.

From the Cassandra stress tool, the scenario is passed to the seed node where in equal reads, writes, number of threads and the experimental duration are constant in all of the experimentations performed.

RQ2) How do the replication factors and consistency level impact the auto scaling of Cassandra Virtual data centers?

The replication factors and consistency levels have shown significant impact on autoscaling of Cassandra nodes. There is an observed change in the latency and CPU utilization values after autoscaling with respect to different replication factors and consistency levels.

- For the three-node cluster scaling over to five nodes, the default configuration of RF1 and CL1 has shown a decrease in overall mean CPU utilization of 7%
- For the configuration RF2 and CL1 has shown a difference of 8% in the overall mean CPU utilization.
- The configuration of RF3CL1 and RF3CLQ have shown relatively higher which is approximately 9% and 8% respectively.

For the five-node cluster scaling over to seven nodes, the following describes the difference in CPU utilization

- For the five-node cluster scaling over to seven nodes, the default configuration of RF1 and CL1 has shown a decreased CPU Utilization of 9%.
- For the configuration RF3 and CL1, the decrement in the overall mean CPU utilization is 3%.
- For RF3 and CLQ configuration, the decrease in the overall mean CPU utilization is 7% .
- For the RF5 and CLQ configuration the mean CPU utilization has decreased by only 2%.

5.2.2 Validity Threats

Following are the possible threats to this thesis study. The following threats are classified as Internal, External and Constructive threats.

5.2.2.1 Internal Threats

Internal threats are the type of threats that the researcher does not acknowledge. These threats usually occur because there has been a problem in the very tools used in the experiment or the design required for the experimentation itself. It is advisable strongly that the errors from the previous research should not be recurring. The efficient way to avoid these types of threats is to repeat the experimentation iteratively for multiple times and the overall mean values for the case of before autoscaling and after autoscaling are considered as the metrics [35]. Therefore, in this case, the experiment to find the threshold has been repeated 5 times and the experiment that determines the latency in joining the Cassandra nodes to EC2 instances has been repeated for another 5 times. Therefore, this threat has been mitigated.

5.2.2.2 External Threats

The results of this experimentation should be generic, meaning that these results should be applicable to other environments. This is considered as an external threat. Even if the results from this experimentation cannot be made generic, it can correspond to the results that would be valid for other conceptual models, applications, and environments [35]. To put this into this thesis' context, cloud providers, database and the tools used in this study can be the major threats as they might be outdated. But, for this research, AWS, Cassandra and Cassandra - stress has been used and all the aforementioned are trending tools and technologies. Therefore, this threat has also been mitigated.

6 CONCLUSION AND FUTURE WORK

This chapter discusses the summary and the conclusions of this thesis study. Furthermore, the extension of the scope of this research is discussed in the future work

6.1 Conclusions

The goal of this research is to quantify the latency in adding Cassandra nodes in Amazon EC2 instances and evaluating the scalability of Cassandra and then putting forth how replication factors and consistency levels have impacted the autoscaling of Cassandra instances.

Primarily the experimentation starts with finding a threshold value as an indicator for AWS to raise the alarm. The threshold values of each of the configurations which are the mean values of the CPU utilizations combined in the cluster are found out. This has been done without autoscaling and without the involvement of AWS itself.

Next, AWS EC2 infrastructure is configured in such a way that upon reaching the threshold value in the next run, AWS launches two new nodes that communicate with the existing 3-node cluster. The experiment is kicked-off with populating the cluster under test with 10 million writes, equal reads and writes and constantly maintaining 609 threads at which performance is noticed to be maximum [27]. Once the data is written into the tables and keyspaces, Cassandra stress commands are used to stress test the Cassandra cluster. The same experiment has been repeated with the 5-node cluster to analyze the trend with increased cluster size.

After reaching the threshold value of mean CPU utilization, Cassandra nodes are autoscaled resulting in data yielding for each specific configuration. The experimentation has been repeated 5 times each since AWS follows pay-as-you-go model.

The mean latency and CPU utilization values are averaged for a three-node cluster and averaged over a five-node cluster separately and have been compared. The results yielded have significantly shown the decrease in latency and CPU utilization for each configuration. For a five-node cluster, the mean latency and CPU utilization are averaged and when scaled the metrics have been averaged over seven-nodes separately.

6.2 Future Work

For future work, the following is proposed.

- To get information from monitoring by using a linear quadratic prediction model and this model will give an info from of what will be in the future of CPU utilization on the basis of previous value.
- Initially, the database on the cluster has been populated with 10 million records with equal reads and writes (50%-50%). Different operations with different mixed ratios can be implemented.
- To collect the data available for the utilization and developing a script that can compute the predicted value and activate Cassandra nodes based on this.
- To explore different configurations and Cassandra specific configurations. For example, this research has only tested this experimentation set-up on 3 and 5 nodes.
- We still do not know the trend on 7 node clusters with different reads, writes, RFs and CLs. Further, as we have considered average CPU utilization as a metric to autoscale, there is disk utilization as a metric to autoscale as well.

- To try the same set up on a different cloud-based platform, for example Openstack or Microsoft azure in order to determine the scalability with the respective autoscaling mechanism.
- As Cassandra does not have scaling mechanism, using AWS step scaling policy to predict when to allocate and start the Cassandra nodes.
- The easiest way is to set up multiple thresholds to breach. When one threshold is passed, the virtual machine can start spinning and when the utilization reaches its actual threshold Cassandra in the instance can start in the foreground and bootstrap Cassandra to the cluster.

REFERENCES

[1] R. Buyya, C. Vecchiola, and S. T. Selvi, *Mastering cloud computing: foundations and applications programming*. Newnes, 2013.

[2] "Whitepapers – Amazon Web Services (AWS)," *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/whitepapers/. [Accessed: 10-Jan-2018].

[3] "Amazon EC2 Instance Types – Amazon Web Services (AWS)," *Amazon Web Services, Inc.* [Online]. Available: //aws.amazon.com/ec2/instance-types/. [Accessed: 07-Feb-2018].

[4] "Launching Auto Scaling Instances in a VPC - Auto Scaling." [Online]. Available: https://docs.aws.amazon.com/autoscaling/latest/userguide/asg-in-vpc.html. [Accessed: 07-Jan-2018].

[5] "Creating a Launch Configuration Using an EC2 Instance - Auto Scaling." [Online]. Available: https://docs.aws.amazon.com/autoscaling/latest/userguide/create-lc-with-instanceID.html. [Accessed: 07-Jan-2018].

[6] "Amazon EC2 Security Groups for Linux Instances - Amazon Elastic Compute Cloud." [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-network-security.html. [Accessed: 07-Jan-2018].

[7] "Simple and Step Scaling Policies - Auto Scaling." [Online]. Available: https://docs.aws.amazon.com/autoscaling/latest/userguide/as-scaling-simple-step.html. [Accessed: 07-Jan-2018].

[8] "Auto Scaling Groups - Auto Scaling." [Online]. Available: https://docs.aws.amazon.com/autoscaling/latest/userguide/AutoScalingGroup.html. [Accessed: 07-Jan-2018].

[9] H. Wang, J. Li, H. Zhang, and Y. Zhou, "Benchmarking replication and consistency strategies in cloud serving databases: Hbase and cassandra," in *Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware*, 2014, pp. 71–82.

[10] "A federated approach on heterogeneous NoSQL data stores - Google Scholar." [Online]. Available: https://scholar.google.se/scholar?hl=en&as_sdt=0%2C5&q=A+federated+approach+on+heterogeneous+NoSQL+data+stores&btnG=. [Accessed: 08-Jan-2018].

[11] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *SIGOPS Oper Syst Rev*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[12] E. Casalicchio, L. Lundberg, and S. Shirinbab, "Energy-Aware Adaptation in Managed Cassandra Datacenters," in *2016 International Conference on Cloud and Autonomic Computing (ICCAC)*, 2016, pp. 60–71.

[13] "Data replication | Apache Cassandraâ„¢ 2.1 | Apache Cassandraâ„¢ 2.1." [Online]. Available: http://docs.datastax.com/en/cassandra/2.1/cassandra/architecture/architectureDataDistributeReplication_c.html. [Accessed: 08-Jan-2018].

[14] "CQL | Apache Cassandraâ„¢ 2.1 | Apache Cassandraâ„¢ 2.1." [Online]. Available: http://docs.datastax.com/en/cassandra/2.1/cassandra/cql.html. [Accessed: 08-Jan-2018].

[15] "The cassandra-stress tool | Apache Cassandraâ„¢ 3.0 | Apache Cassandraâ„¢ 3.0." [Online]. Available: https://docs.datastax.com/en/cassandra/3.0/cassandra/tools/toolsCStress.html. [Accessed: 08-Jan-2018].

[16] T. Debbarma and K. Chandrasekaran, "Comparison of FOSS based profiling tools in Linux operating system environment," in *Contemporary Computing and Informatics (IC3I), 2016 2nd International Conference on*, 2016, pp. 65–72.

[17] J. Kuhlenkamp, M. Klems, and O. Röss, “Benchmarking scalability and elasticity of distributed database systems,” *Proc. VLDB Endow.*, vol. 7, no. 12, pp. 1219–1230, 2014.
[18] V. Abramova, J. Bernardino, and P. Furtado, “Testing Cloud Benchmark Scalability with Cassandra,” in *2014 IEEE World Congress on Services*, 2014, pp. 434–441.
[19] A. Gandini, M. Gribaudo, W. J. Knottenbelt, R. Osman, and P. Piazzolla, “Performance Evaluation of NoSQL Databases,” in *Computer Performance Engineering*, A. Horváth and K. Wolter, Eds. Springer International Publishing, 2014, pp. 16–29.
[20] B. G. Tudorica and C. Bucur, “A comparison between several NoSQL databases with comments and notes,” in *2011 RoEduNet International Conference 10th Edition: Networking in Education and Research*, 2011, pp. 1–5.
[21] Y. Li and S. Manoharan, “A performance comparison of SQL and NoSQL databases,” in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 15–19.
[22] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser, “Application-Specific Evaluation of No SQL Databases,” in *2015 IEEE International Congress on Big Data*, 2015, pp. 526–534.
[23] S. P. Kumar, S. Lefebvre, R. Chiky, and E. G. Soudan, “Evaluating consistency on the fly using YCSB,” in *2014 International Workshop on Computational Intelligence for Multimedia Understanding (IWCIM)*, 2014, pp. 1–6.
[24] M. Chalkiadaki and K. Magoutis, “Managing Service Performance in the Cassandra Distributed Storage System,” in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013, vol. 1, pp. 64–71.
[25] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai, “Optimal autoscaling in a IaaS cloud,” in *Proceedings of the 9th international conference on Autonomic computing*, 2012, pp. 173–178.
[26] Yair Levy and Timothy J. Ellis, “A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research.” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.2369&rep=rep1&type=pdf. [Accessed: 17-Mar-2017].
[27] J. Rowley and F. Slack, “Conducting a literature review,” *Manag. Res. News*, vol. 27, no. 6, pp. 31–39, 2004.
[28] “DataStax: always-on data platform | NoSQL | Apache Cassandra.” [Online]. Available: https://www.datastax.com/. [Accessed: 10-Jan-2018].
[29] Florian Fittkau, “Controlled Experiments in Software Engineering.”
[30] C. Wohlin, M. Höst, and K. Henningsson, “Empirical research methods in software engineering,” in *Empirical methods and studies in software engineering*, Springer, 2003, pp. 7–23.
[31] “Apache Cassandra.” [Online]. Available: http://cassandra.apache.org/. [Accessed: 07-Jan-2018].
[32] H. Cooper and L. V. Hedges, *The Handbook of Research Synthesis*. Russell Sage Foundation, 1993.
[33] E. Tang and Y. Fan, “Performance Comparison between Five NoSQL Databases,” in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, 2016, pp. 105–109.
[34] K. R. Jackson *et al.*, “Performance analysis of high performance computing applications on the amazon web services cloud,” in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, 2010, pp. 159–168.
[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.