

Thesis no: MSEE-2016-21



Compactions in Apache Cassandra

Performance Analysis of Compaction Strategies in Apache Cassandra

Srinand Kona

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 Masters in Electrical Engineering with emphasis on Telecommunication Systems. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:

Author:

Srinand Kona

E-mail: srko15@student.bth.se

External advisor:

Jim Hakansson

Ericsson

Karlskrona, Sweden

University advisor:

Dr. Emiliano Casalicchio

Computer Science and Engineering Department

Blekinge Institute of Technology

Kalskrona, Sweden

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. The global communication system is in a tremendous growth, leading to wide range of data generation. The Telecom operators in various Telecom Industries, that generate large amount of data has a need to manage these data efficiently. As the technology involved in the database management systems is increasing, there is a remarkable growth of NoSQL databases in the 20th century. Apache Cassandra is an advanced NoSQL database system, which is popular for handling semi-structured and unstructured format of Big Data. Cassandra has an effective way of compressing data by using different compaction strategies. This research is focused on analyzing the performances of different compaction strategies in different use cases for default Cassandra stress model. The analysis can suggest better usage of compaction strategies in Cassandra, for a write heavy workload.

Objectives. In this study, we investigate the appropriate performance metrics to evaluate the performance of compaction strategies. We provide the detailed analysis of Size Tiered Compaction Strategy, Date Tiered Compaction Strategy, and Leveled Compaction Strategy for a write heavy (90/10) work load, using default cassandra stress tool.

Methods. A detailed literature research has been conducted to study the NoSQL databases, and the working of different compaction strategies in Apache Cassandra. The performances metrics are considered by the understanding of the literature research conducted, and considering the opinions of supervisors and Ericsson's Apache Cassandra team. Two different tools were developed for collecting the performances of the considered metrics. The first tool was developed using Jython scripting language to collect the cassandra metrics, and the second tool was developed using python scripting language to collect the Operating System metrics. The graphs have been generated in Microsoft Excel, using the values obtained from the scripts.

Results. Date Tiered Compaction Strategy and Size Tiered Compaction strategy showed more or less similar behaviour during the stress tests conducted. Level Tiered Compaction strategy has showed some remarkable results that effected the system performance, as compared to date tiered compaction and size tiered compaction strategies. Date tiered compaction strategy does not perform well for default cassandra stress model. Size tiered compaction can be preferred for default cassandra stress model, but not considerable for big data.

Conclusions. With a detailed analysis and logical comparison of metrics, we finally conclude that Level Tiered Compaction Strategy performs better for a write heavy (90/10) workload while using default cassandra stress model, as compared to size tiered compaction and date tiered compaction strategies.

Keywords: Apache Cassandra, Compaction Strategies, Default Cassandra Stress model, Performance.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Emiliano Casalicchio, for his continuous support and guidance to achieve the aim of this project.

I sincerely thank Prof. Lars Lundberg for his support in bringing this project in to a good shape in the initial times of this thesis.

I would like to thank Julia Sidorova, for helping us in providing this thesis topic.

I would also like to thank Christine Niyizamwiyitira, and Sogand Shirinbab, for their help and continuous support in technical issues. I thank BTH IT Department, and Christine Niyizamwiyitira for providing the bare metal server.

I express my sincere gratitude to Jim Hakansson at Ericsson, for supervising us from the Industry side. I would also like to thank Marcus Olsson, Per Otterström, Mattias Carlsson C, who work at Ericsson, for their valuable advices and continuous support for this project.

I would like to thank Swaroopa Ravu for her continuous support and partnering for this Master thesis project.

LIST OF FIGURES

- Figure 2-1: Cassandra Cluster Ring Architecture.....10
- Figure 5-1: CPU User, CPU Privileged, CPU Idle for LCS, DTCS and STCS.....20
- Figure 5-2: Operations/second for LCS, DTCS and STCS.21
- Figure 5-3: Latency Mean for LCS, DTCS, and STCS.....22
- Figure 5-4: Total Compactions for LCS, DTCS, and STCS.22
- Figure 5-5: Disk Usage Percent for LCS, DTCS, and STCS23
- Figure 5-6: Disk Read Rate.for LCS, DTCS, and STCS24
- Figure 5-7: Total Bytes Compacted for LCS, DTCS, and STCS.25
- Figure 5-8: Memory Cached for LCS, DTCS, and STCS.25

LIST OF TABLES

Table 1: Operations/second.	21
Table 2: Total Compactions.....	23
Table 3: Total Disk Reads.	24

CONTENTS

ABSTRACT	I
ACKNOWLEDGEMENTS	II
LIST OF FIGURES	III
LIST OF TABLES	IV
CONTENTS	5
1 INTRODUCTION	6
1.1 OBJECTIVES	7
1.2 RESEARCH QUESTIONS	7
1.3 DOCUMENT OUTLINE	7
1.4 SPLIT OF WORK	8
2 BACKGROUND	10
2.1.1 <i>Architecture</i>	<i>10</i>
2.1.2 <i>Data Model</i>	<i>10</i>
2.1.3 <i>Compaction</i>	<i>11</i>
3 RELATED WORK	13
4 METHODOLOGY	15
4.1 TEST BED	15
4.2 METRICS	15
4.3 WORKLOADS	16
4.4 CLUSTER CONFIGURATION	17
4.5 STRESS COMMAND	17
4.6 EXPERIMENT METHOD	18
5 RESULTS AND ANALYSIS	20
6 CONCLUSION AND FUTURE WORK	26
REFERENCES	28

1 INTRODUCTION

As the global communication system is growing rapidly, there is a tremendous increase in data generation. The operators in various Telecom Industries which generate huge amount of data, have to manage these data efficiently. These data include information that is to be monitored regularly, and also the information that is not monitored regularly, but have to be saved for future purposes. There are various techniques developed in managing this big data. Databases play a serious role to store and manage these data efficiently. Relational Database Management Systems (RDBMS), and NoSQL[1] databases are the mostly used databases by almost every company which has a need to store the data. Relational databases store the data that is structured and assigned in a specific format, where as NoSQL databases can store unstructured or semi-structured data. NoSQL is widely being used to manage big data, as the data can be directly dumped into the database without any modification. There are many NoSQL database management systems available, and Apache Cassandra is the popular among all.

Apache Cassandra is an open-source decentralized or distributed database management system which is designed to handle large amounts of semi-structured and unstructured formats of data across many servers providing scalability and high availability, without compromising in its performance. Apache Cassandra was initially developed at Facebook, to power their inbox search feature. Cassandra's innate datacenter concepts are important as they allow multiple workloads to be run across multiple datacenters [1]. Cassandra is developed as a NoSQL database management system that is designed unlike the tabular relations used in relational databases or traditional databases, which results in simple design and horizontal scaling to clusters of machines. They have an increasing demand in big data and real time web applications. Apache Cassandra is designed by the Apache software. Cassandra is being used by various large companies in different platforms like, Ericsson, Facebook, Google, Netflix, Portugal Telecom to manage the large amounts of data generated by their operations. Cassandra uses Cassandra Query Language (CQL) for the primary interface of the Cassandra Database, which looks and acts like SQL[2].

Different compaction strategies[3] are developed in Apache Cassandra to provide better performance of the system, by reducing the unnecessary data generated, and reducing the disk space through compaction. Until now, there are three types of compaction strategies, Size Tiered Compaction strategy, Leveled Compaction Strategy and Date Tiered Compaction Strategy. The selection of the compaction strategy in an operation is done manually. This paper focuses on providing the performances of different compaction strategies in different use cases, in order to decide the right type of compaction strategy to use. This thesis solves a problem at Ericsson, which is to find the right compaction strategy for write heavy workload at different stages of operation[2].

Management of the huge volumes of data is a major need for any large company. Though Apache Cassandra is a solution to efficiently manage these data, there is still a need to develop the technicalities and find a better way to use it effectively[4]. In Cassandra, the compaction strategy should be selected manually. But, as there can be many use cases, one cannot know which type of compaction strategy to select for a specific use case.

The thesis discusses different types of compaction strategies in Apache Cassandra[5], and their performances for write heavy workload using default Cassandra stress model, while considering the useful metrics. The main idea is to provide a proper

selection of compaction strategy for Write heavy workload using default cassandra stress model.

1.1 Objectives

The main objectives of this thesis are specified in this chapter.

1. Investigate the working and behaviour of size tiered compaction strategy, leveled compaction strategy and date tiered compaction strategy in Apache Cassandra.
2. Identify the appropriate metrics that are to be observed to evaluate the performance of compaction strategies for a write heavy workload, using default cassandra stress model.
3. Conduct experiments to measure the performances of size tiered compaction strategy, leveled compaction strategy, and date tiered compaction strategy for write heavy workload using default cassandra stress model.
4. Analyze and evaluate the data obtained from the experiments to document the performances of compaction strategies in Apache Cassandra for write heavy workload using default cassandra stress model, in order to suggest a better working compaction strategy for a write heavy data.

1.2 Research Questions

A few research questions are framed for this thesis, and they are classified in this section.

1. What are the performance metrics to be observed to evaluate the success of a compaction strategy for a write heavy data?
The performance metrics include Cassandra metrics and OS metrics, which are selected by considering the opinions of the advisors of this project, which are discussed in the Chapter 4.2
2. How to measure the performances of different compaction strategies of Cassandra using default Cassandra stress model for Write heavy (90/10) workload?
Two different tools are developed to collect the performances of the compaction metrics and OS metrics while the experiments for different compaction strategies are carried out.
3. Which compaction strategy is appropriate for a write heavy (90/10) workload, using default cassandra stress model?
After collecting the unbiased values from the experiments conducted, and a detailed analysis, it is concluded that Leveled Compaction Strategy is appropriate for a write heavy workload while using default cassandra stress model.

1.3 Document Outline

This document is organized as mentioned below:

Chapter 1 gives the overview of the current trend in NoSQL Databases, and explains about Apache Cassandra database and also gives a view on compaction strategies in Cassandra. This chapter exposes the current usage of Cassandra, which is a

popular NoSQL database. The main problem associated with Cassandra database, which motivated to select this project is mentioned. The outline of the thesis work done, and the main focus of this thesis work is discussed in this chapter. The main objectives and a few research questions that are framed for this thesis are mentioned.

Chapter 2 discusses the technical details of Apache Cassandra, and its compaction strategies. The architecture of Cassandra is clearly explained. The data model of the cassandra database, and default cassandra stress model is described in this section. The working of different compaction strategies in Cassandra is clearly elaborated.

Chapter 3 is about the related work done on Cassandra databases. It is written based on the literature review conducted. This chapter discusses the various projects that are conducted on cassandra.

Chapter 4 is about the methods that are used to conduct the experiments. This section discusses about the experiment environment, metrics that are considered, workloads used, cluster configuration, stress command used to generate the stress data, and the procedure for the experiments conducted.

Chapter 5 shows the results that are obtained from the experiments conducted in this project, and a detailed analysis of them. The results are graphical representation and in tabular forms. The analysis of the results in this chapter is done by logically comparing different parts of the results.

Chapter 6 gives a conclusion derived from the results and the analysis. This part describes the work done to complete the thesis. It also motivates the conclusion statement that is derived from the results.

1.4 Split of Work

This thesis is a part of a big project at Ericsson, which is to analyze the performances of all the compaction strategies in Apache Cassandra, for write heavy (90/10), read heavy (10/90), Balanced (50/50) workloads. The project is divided within two persons, Srinand Kona, and Swaroopa Ravu, based on the workloads. Write heavy workload is opted for Srinand, and Read heavy and Balanced workloads are shared for Swaroopa. The test bed is same for both the persons. Experiments are conducted individually for the respective workloads.

Some parts of this document involves the contribution of both Srinand and Swaroopa, and some are individual contributions.

Chapter 1, which involves overview of cassandra, motivation, objectives, and research questions are written by Srinand, by his understanding on the work involved in this thesis.

Srinand and Swaroopa has done the literature research together, and hence the Background in Chapter 2 is a contribution of both. In Chapter 2, the architecture, data model, and description of default cassandra stress model is contributed by Swaroopa. And, other part in Chapter 2, which involves the detailed description of compaction strategies is written by Srinand.

Chapter 3, related work, is written by both Srinand and Swaroopa based on their literature research. This chapter discusses the description of individual projects done on Cassandra.

Chapter 4, which involves the description of the test bed is written by Swaroopa. The different metrics that are finalized for the experimentation by considering the opinions of the advisors of this project, are written by both Srinand and Swaroopa. The discussion of workload involved in this thesis, cluster configuration, elaboration of stress command, and the procedure for the experiments conducted is written by Srinand. The two different tools that are used for collecting the performances during the experiments is developed by Srinand.

The graphs for write heavy workloads are created by Srinand using Microsoft Excel, which are presented in results in Chapter 5. The logical reasoning and the analysis is also done by Srinand.

The conclusion after a detailed analysis, is derived by Srinand. The future work is also done by Srinand by his understanding on the future scope related to this thesis.

2 BACKGROUND

2.1.1 Architecture

Apache Cassandra architecture can be visualized as a ring with multiple number of nodes, communicating with each other, which is represented in Figure 2-1. The main design goal of Cassandra is to handle heavy data workloads across multiple nodes, without any single point of failure. Cassandra has a peer-to-peer decentralized and distributed system among all the nodes in a cluster[6]. Each node in a Cassandra cluster is interconnected to other nodes, and all the nodes play the same role. Regardless on which node the data is located, each node can accept the read and write requests. When a node fails, the requests can be served from other nodes in the cluster[7]. The architecture is developed by understanding that the system or the hardware can fail at any instant. Each node exchanges the information across the cluster every second. Any authorized user can connect to any node in any datacenter, and can access the data using CQL. There are no master nodes in a Cassandra cluster. When a client connects to a node in a cluster that node coordinates the cluster according to the requests. Cassandra uses Gossip Protocol which is a peer-to-peer communication protocol to share the location information of other nodes, in the background, to allow the nodes to communicate with each other[8].

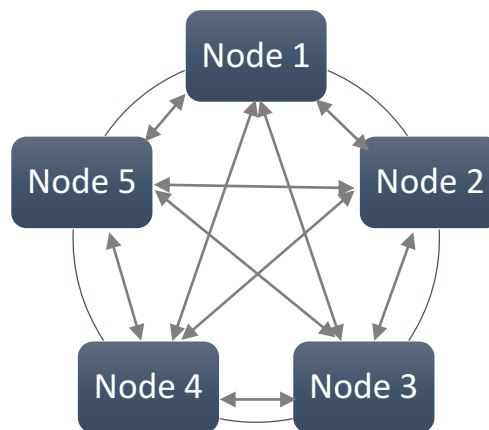


Figure 2-1: Cassandra Cluster Ring Architecture.

2.1.2 Data Model

Cassandra database is distributed over sever nodes in the cluster. Every node has a replica, and if any node fails, the replica can be taken in to charge. The data model of Cassandra is different from that of Relational databases, as it uses the storage structure similar to Log-Structured Merge Tree. As the storage structure of Cassandra is a log-structured engine, it avoids overwrites of data and uses sequential IO to update data. Cassandra never re-writes or re-reads the existing data, and also never overwrites the rows [9]. The data in Cassandra is stored in a container called Keyspace, which is the outermost layer. Cassandra has a flexible schema and deals with unstructured data

Keyspace involves a few basic attributes called Replication Factor, Replica placement strategy, Column families[4]. Replication factor is a number of machines in the cluster that have to receive the copies of the data. Cassandra follows a strategy for the replication of the data in the node ring, which is called as Replica Placement Strategy. There are different kinds of strategies such as simple strategy (rack-aware

strategy), old network topology strategy (rack-aware strategy), and network topology strategy (datacenter-shared strategy). A table in Cassandra database is a list of nested key-value pairs (Row \times Column Key \times Column value)[10].

Column families represent the structure of the data. Keyspace contains one or more column families. Column families contains ordered collection of rows, and each row contains ordered collection of columns. In Cassandra, we can freely add any column to any column family at any time. The column families are defined, and the columns are not defined. Column is a basic data structure of Cassandra with values key or column name, value, and time stamp.

Cassandra processes data at different stages. The first stage of processing the data is logging data in the commit log. When a write request is generated, the data is stored in a structure in a memory called memtable, and logs the data on disk which thus provides durability. This commit log receives the data whenever a write request is generated. The second stage in processing the data involves flushing the data from the memtable. When the data in the memtable storage exceeds a specific configurable threshold, then the data is put in a queue to be flushed to the disk. The queue size can be configured manually, and if the data to be flushed exceeds the queue size, then the write requests are blocked until the data to be flushed succeeds. The flushing of data can also be done manually by using nodetool utility. The third stage is to store the data on disk in SSTables. After the data is flushed, Cassandra writes the data into disk sequentially in SSTables. SSTable is a Sorted String Table, which is a file of key or value string pairs that are sorted by keys. SSTable file has three separate files created, and they are per column-family. The three separate files are Bloom Filter, Index and Data[11]. The final stage of processing the data in Cassandra is compaction. The compaction in Cassandra takes place in the SSTables. Compaction is the process of merging the data by using several specified techniques.

2.1.2.1 Default cassandra stress model

Default cassandra stress tool is a stress tool that is available in the cassandra package, by default. The cassandra stress tool is a java based stress testing utility for basic benchmarking and load testing a cassandra cluster . It defines the type of data model for a specific workload. The stress tool populates the cluster according to the given specifications. It is used to determine the performance of cluster for specific use cases. It also helps in understanding the working of the database. The Cassandra stress tool supports YAML based profiles for defining the data model, in which compaction strategies, cache settings, keyspace, and settings can be performed. The default Cassandra stress tool creates a keyspace called '*keyspace1*', and within that, it creates tables named as '*standard1*' and '*counter1*'. These are automatically created when the stress test is run, and can be reused for several iterations. We can also specify the type of workload in the stress command. Different types of compaction strategies can be set by changing the name of the compaction strategy in the specified YAML[12] profile.

2.1.3 Compaction

In Cassandra compaction is the process of merging keys, combining columns, deleting tombstones, consolidating SSTables, and creating a new index in the merged SSTable [3]. Compaction merges data in each SSTables by selecting the storage data according to the timestamp. Compaction techniques combine multiple SSTables into single SSTable in order to increase the performance. There are different types of compactions introduced, as the Cassandra grows its versions. Until now, there are three types of compaction strategies, Size tiered compaction strategy, Leveled Compaction

Strategy, and Date Tiered compaction strategy. Only one compaction strategy can run at a time for a specific operation.

2.1.3.1 Size Tiered Compaction Strategy (STCS)

Size Tiered Compaction strategy is the oldest compaction strategy, and it is still the default compaction. In this strategy, the compaction is done when the disk has specific number (four by default) of SSTables with similar size. These small SSTables are compacted together to form a single large SSTable. These already compacted tables can be compacted further with the similar size of tables to form even larger SSTables, and this process continues until the compaction ends. Hence, the system has data in various tiers of tables. When one tier is full, then all the tables in that tier are merged to form a single table. The reads during this compaction doesn't work well as there can be many versions of rows in different SSTables. Hence, this compaction can work better if the data is only once generated and not modified later.

2.1.3.2 Leveled Compaction Strategy (LCS)

Leveled compaction strategy was the second introduced compaction strategy of Cassandra. In this compaction strategy, the system considers small fixed size (160MB by default) of SSTables divided into different levels. In the first level, level 0, the new SSTables are created in which the flushed data from memtables is present. In the levels other than level 0, they are exponentially increasing in size. The data in these SSTables is sorted by Log-structured merge-tree data structure. Firstly, the SSTables in level 0 are compacted with the SSTables in level 1, so that new SSTables are formed with compacted data of fixed size (160MB by default) each. This results in extra SSTables in level 1. These extra SSTables are compacted with the SSTables of level 2 again to form new SSTables of fixed size (160MB by default) each. The process continues until the end of all the levels.

2.1.3.3 Date Tiered Compaction Strategy (DTCS)

Date tiered compaction strategy is currently the latest compaction strategy developed and designed for the time series data, but also can be used for other kinds of data as well. In the time series data the data is inserted in an organized way, according to timestamp. In date tiered compaction strategy, the system sorts the SSTables according to time, and then compacts the adjacent SSTables. The default base time seconds for one SSTables is 60 seconds, and the default minimum threshold value for number of SSTables to compact is 4. The date tiered compaction strategy works similar to size tiered compaction strategy. When 4 small one minute SSTables are compacted, a 4 minute SSTable is formed. This 4 minute SSTable compacts with other three 4 minute SSTables to form a 16 minute SSTable. This process continues until the end of the data. SSTables that are more than fixed maximum time period (365 days by default) are deleted, as they may reduce the performance of the system, and cannot perform read and write requests successfully.

3 RELATED WORK

There is a lot of research done on the NoSQL systems, Apache Cassandra, the benefits and flaws of using Cassandra, and other research works or projects done on Apache Cassandra to obtain the required knowledge required for this project.

In paper[13] “Evaluating NoSQL Technologies for Historical Financial Data”, the author provides an efficient solution for storing and processing huge volume of data for certain variants. The author discusses the advantages of NoSQL databases over Relational Database Management Systems (RDBMS) and explore various NoSQL databases at Cinnober Financial Technology AB. The main aim of this paper is to compare the NoSQL databases Apache Cassandra and Apache Lucene and traditional relational database MySQL, for financial management. The author concludes that Apache Cassandra is an efficient solution for storing and processing huge amounts of data with different variants. The also conveys that the Apache Cassandra’s performance does not get affected even if the amount of data in the database is tremendously increased. This research motivated us to select Cassandra database, and work on its details.

The thesis [14] the evaluation of non-functional properties of NoSQL database management systems is done. The author preferred to compare MongoDB, Redis, and Apache Cassandra, by considering the NoSQL properties, Document Stores, Key Value Stores and Column Stores. The author used YSCB benchmarking tool to perform the evaluation. The author concluded by saying that Apache Cassandra is the best choice to handle an enormous amount of data. Based on the conclusion of this thesis paper, large amount of stress data is generated for the experiments.

In [15]the author performs a study to investigate the performance of serialization using Apache Avro library in Cassandra database. In this paper, different serialized data models are compared with traditional relational database model. This study involves an experimental approach that compares read and write latencies using Twitter data in JSON format. The results of the experimental study of this paper says that Avro serialization can improve the performance, and the performance is highly dependent on the serialization granularity defined by the data model. The author concludes that developers seeking to improve database throughput in Apache Cassandra through serialization should prioritize data model optimization as serialization by itself will not outperform relational modelling in all use cases. This helped us to investigate different types of important workloads and data models, that can be used for the experiments.

Thosen Sofie, the author of “Replica Selection in Apache Cassandra” [16], focuses on reducing the tail latency for reads in the Apache Cassandra database system, by implementing the new replica selection algorithm C3, which was developed by Lalith Suresh, Marco Canini, Stefan Schmid and Anja Feldmann. The author found in his study that C3 decreases the tail latencies of generated load through extensive benchmarks, but in the case of production load, the results does not show any particular improvement. The author claims the reason for this is mostly due to the variable size records in the data set and token awareness in the production client. The author finally concludes that the server side C3 algorithm will work effectively for the systems with homogeneous row sizes where the clients are not token aware. This helped us to understand about the datasets in Apache Cassandra.

The paper “Large-scale analysis and storage of raw data in the cloud”[17], the authors have tried to evaluate Apache Cassandra and Apache Hadoop based on the storage and usage of large amounts of unstructured log data. They finalized that the

combination of these two software are complementary and provides a wide range of functionality. Although each individually performs their task the best, Cassandra has high reliability and fast write performance making it suitable for storage of streaming log data. This paper have motivated us to select Cassandra database for this thesis.

A study on Ericsson's voucher system and its Cassandra cluster is conducted to identify the different Cassandra specific and product specific factors affecting the disk space in Cassandra data storage system in[8]. In this study a model is built to predict the disk storage of Ericsson's voucher system. The model is built based on specific interviews and research, and using this model the factors affecting disk space for deploying Cassandra are identified to make the capacity planning process more efficient. This study made us aware about different factors that affect disk space for deploying Cassandra, and also helped in predicting the analysis for disk usage and disk utilization.

In the Thesis [18], the author tries to explain the need of distributed databases for Multi Mediation processes. The paper aims at analyzing the distributed databases, MySQL Cluster 7.4.4 and Apache Cassandra 2.0.13, in terms of performance, scalability, and availability. The research involves qualitative and quantitative study. A benchmarking harness application is designed to evaluate the performance of distributed databases in the context of Multi Mediation. Several experiments are conducted using the benchmarking harness application on the database cluster. The results of the experiments conducted say that MySQL cluster average select response time is better than Apache Cassandra for greater number of client threads in high availability and low availability. The author further concludes that Apache Cassandra outperforms MySQL cluster while considering the support for transaction and ACID compliance in the databases. This helped us to mention the various advantages of NoSQL databases over MySQL databases in the paper.

In 2012, University of Toronto researchers studying NoSQL systems concluded that, Apache Cassandra is a clear winner, as it achieves the highest throughput for the maximum number of nodes in all experiments considering the cases of high write latency and high read latency [13]. This helped us to understand the benefits of using Apache Cassandra, and also helped in developing the background knowledge on Cassandra.

All the studies conducted on Apache Cassandra shows that Cassandra is best for its way of handling huge amounts of unstructured as well as structured data. It has high performance capabilities, high availability, and scalability. These studies motivate for further research on Apache Cassandra and its functionalities in deeper.

4 METHODOLOGY

The Methodology of this Thesis work includes, detailed literature research, finding the appropriate performance metrics that are needed to be compared in order to analyze the performance of each compaction strategy. The metrics are decided by taking the opinions of experts in the field, which includes the advisors at Ericsson Karlskrona, and professors at Blekinge Institute of Technology. The method to approach the aim of this project involves an experimentation part from which the required data and graphical representations can be obtained to make an appropriate analysis. The experimentation includes collecting the values of the selected metrics and plotting the graphs for the values obtained.

A tool is developed to collect the values of the selected metrics and store them in to a log file. The tool is developed in jython. Jython is a scripting language, which is an implementation of python on java platform.

4.1 Test Bed

For the experimentation that is involved in this project, a specific environment is created. The experiments are conducted on the Linux server of Ubuntu platform, which is provided by the administration of Blekinge Institute of Technology with SSH access. This is a bare metal server, which has a memory of 266.55GB. The latest Cassandra version release, Cassandra3.5 is installed by tarball installations on the server from apache mirrors. A one node Cassandra cluster is developed, and the tests are performed on that node, keeping in mind that a cluster with more than one node can effect the performances as the operations are dependent. Java8.0 version is installed on this server. Jython and python are also installed on the platform. ‘psutil’ package is installed. The tests are run for the default cassandra stress model, where the stress is generated for all the specified use case. All the configurations and settings are made default while conducting the experiments.

4.2 Metrics

The appropriate metrics that are to be considered for the evaluation of the performances of compaction strategies are finalized after considering the opinions of the experts in the field. The metrics involve OS metrics as well as compaction metrics

- **Write Requests:** Number of write requests per second. Monitoring the number of write requests over a given period of time, reveals system write workload and usage patterns.
- **Read Requests:** Number of read requests per second. Monitoring the number of read requests over a given period of time, reveals system read workload and usage patterns.
- **Write Request Latency:** It is the average response times of a client write. The time period starts when a node receives a client write request, and ends when the node responds back to the client.
- **Read Request Latency:** It is the average response times of a client read requests. The time period starts when a node receives a client read request, and ends when the node responds back to the client.

- **Operations per second:** It is the number of overall operations conducted in a time period. The operations can be write or read or both.
- **Latency Mean:** It is the mean of total round time of a data packet, when it is transmitted and returned back to its source.
- **Total Bytes Compacted:** It is the total size of the compactions performed per second.
- **Total Compactions:** The total number of compaction tasks completed per second.
- **Compactions Pending:** Total number of compactions required to achieve the desired state.
- **Read Requests Completed:** Number of completed read requests at the specific time.
- **Write Requests Completed:** Number of write requests completed at the specific time.
- **Memory Used:** Total system memory used for the conducted task.
- **Memory Cached:** Total system memory cached for the conducted task.
- **OS Load:** The average of load on the operating system.
- **CPU User:** Time that CPU devotes to user processes
- **Disk Usage (%):** The total disk space used by Cassandra for the specific operation performed.
- **Disk Write Throughput:** Average disk throughput (in MB) for write operations at a specific time.
- **Disk Read Throughput:** Average disk throughput (in MB) for read operations at a specific time.
- **Disk Throughput:** Average disk throughput (in MB) for both write and read operations at a specific time.
- **Disk Write Rate:** Number of write requests on disk per second.
- **Disk Read Rate:** Number of read requests on disk per second.
- **Disk Latency:** Average completion time of each request to the disk
- **Disk Utilization:** Total CPU time consumed by disk.

4.3 Workloads

The workload considered for this project is write heavy (90/10). Write heavy is a mixed workload which involves both write and read requests. (90/10) Write heavy

(90/10) is a write heavy workload with 90 percent of writes and 10 percent of reads. The work load is specified in the stress command, so that the stress data is generated accordingly.

4.4 Cluster Configuration

In order to achieve unbiased results of the experiments conducted for this project, a one node cluster is configured for Cassandra 3.5 version. After the installation of Cassandra through tarball installations, the *'cassandra.yaml'* file which exists in the directory *'~/directory of cassandra installed/conf/cassandra.yaml'* is edited as per the requirement. In this case, as all the options are left default, it is checked whether the *'listen address'*, *'rpc address'*, and *'seeds'* are set for *'localhost'* or *'127.0.0.1'*. The node is set up when the cassandra is run by using the command *'bin/cassandra -R'*, in the cassandra main directory. The status of the node is checked by using *'./nodetool status'* command in the *'cassandra/bin'* directory. All the stress tests are conducted on this node, to obtain the values of the selected metrics. To conduct the stress tests and for any operations on the node, the node status should be *'up'*, i.e. Cassandra should be run in the background.

4.5 Stress Command

The stress command invokes the given specifications. It is used to run the cassandra stress tool *'cassandra-stress'* present in the directory *'~/directory of cassandra installed/tools/bin/cassandra-stress'*. Below is an example of the stress command used to generate the data. The command is run in the directory where cassandra is installed.

Stress command: *"sudo ./tools/bin/cassandra-stress user profile= directory of YAML file ops\(\(insert=9, simple1=1\) n=1000000 duration=20m -rate threads=8 -graph file=filename title=titlename revision=compaction type -log file=log file directory/log file name interval=1s"*

- ***'./tools/bin/cassandra-stress'***: It invokes the *'cassandra-stress'* file present in the directory *'/tools/bin/'* in cassandra package, to create a data model.
- ***'user profile=directory of YAML file'***: This part of the command provides the directory of YAML file, which can be used to generate the required stress. In this YAML file, the type of compaction and other related settings can be made.
- ***'ops\(\(insert=9, simple1=1\)'***: ops refer to operations. This is the specification of the workload. It specifies write heavy workload. Where insert is the writes and simple1 is reads according to the YAML file. As the aim of this project is to make tests for write heavy workload, it is specified as insert or writes as 9 and simple1 or reads as 1, which means 90 percent of writes and 10 percent of reads are generated for the entire data.
- ***'n=1000000'***: 'n' is the number of operations to run. It is set to one million to generate heavy amount of data, so that a real environment can be created.
- ***'duration=20m'***: This is to specify the total time to run the command. The stress is generated for 20 minutes, by this specification. The duration is given to 20 minutes so that it can be enough for the compactions to occur.

- **'-rate threads=8'**: This is to specify the number of threads for the operation. The threads are set to '8' so that the maximum stress is obtained without overloading the system.
- **'-graph file=filename'**: The graph is generated for the data generated. The graph shows the latency mean, operations per second, etc. The graph can be used for easily monitoring the operations of the system.
- **'-graph title=titlename'**: It specifies the name of the graph.
- **'revision=compaction type'**: This is to specify the type of compaction strategy.
- **'-log file=log file directory/log file name'**: The log file specification specifies the directory of the file in which the data generated is stored, and the entire process during the command is also stored.
- **'interval=1s'**: This is to mention that the logs should be saved into the file for every 1 second interval.

4.6 Experiment Method

To conduct the experiments in this project, two different tools were developed, to collect the performances for the considered metrics. The tools are written in Jython and python scripting languages. Jython script is used to collect the performances of the metrics that are related to compactions, and other script, which is written in Python, is used to collect the system or OS metrics. The Jython script is a Python script in which Java commands can be executed. It connects to the JMX and MBeans server to collect the instantaneous values for the considered compaction related metrics. The Python script uses 'psutil' library to collect the instantaneous values for system metrics.

After the installation of a one node cluster, the node state should be 'up' to perform any operations on it. The node state goes up if the cassandra is run. Cassandra is to be run by typing the command 'cassandra -R' in the bin directory of cassandra package. We can make sure the node status is 'up' by typing the command 'bin/nodetool status' in the cassandra installed directory.

Three different YAML files 'cqlstress-date.yaml', 'cqlstress-size.yaml', 'cqlstress-level.yaml' are created by replicating the file 'cqlstress-example.yaml' in the tools directory of the cassandra package. These three files are created so that they can be used to make a default cassandra stress data model for three different compaction strategies. The compaction fields in the file are changed to the respective names to invoke that specific compaction strategy when the file is used for the operations.

The experiment environment is created by from the above specifications. Before generating the data through the stress command, the tools that are developed to collect the values of the considered metrics are to be run. The jython script is run by the command 'jython cm.py>CMLogs/size_cm_writeHeavy.csv' where 'cm.py' is name of the Jython file, and the data is saved to a csv file of the specified file name and directory. Similarly, the python script for system metrics is run by the command, 'python os.py>OSLogs/size_os_writeHeavy.csv' where 'os.py' is the name of the python file, and the data is saved into a csv file of the specified filename and directory in the command.

After the scripts are running, the stress tool has to be run, to generate the desired stress data.

The stress commands used for different compaction strategies are:

Leveled Compaction Strategy: “*sudo ./tools/bin/cassandra-stress user profile=tools/cqlstress-level.yaml ops\{insert=9, simple1=1\} n=1000000 duration=20m -rate threads=8 -graph file=write-dev-benchmark.html title=Write revision=LeveledCompaction -log file=graphsLogs/level_writeHeavy.log interval=1s*”

Size Tiered Compaction Strategy: “*sudo ./tools/bin/cassandra-stress user profile=tools/cqlstress-size.yaml ops\{insert=9, simple1=1\} n=1000000 duration=20m -rate threads=8 -graph file=write-dev-benchmark.html title=Write revision=SizeTieredCompaction -log file=graphsLogs/size_writeHeavy.log interval=1s*”

Date Tiered Compaction Strategy: “*sudo ./tools/bin/cassandra-stress user profile=tools/cqlstress-date.yaml ops\{insert=9, simple1=1\} n=1000000 duration=20m -rate threads=8 -graph file=write-dev-benchmark.html title=Write revision=DateTieredCompaction -log file=graphsLogs/date_writeHeavy.log interval=1s*”

After completing all the experiments, the graphs are generated using Microsoft Excel by using the csv files where the data of compaction metrics and the system metrics are collected. The data of operations/second and latency mean metrics are used from the log files created through the stress command.

5 RESULTS AND ANALYSIS

The results of the experiments conducted are discussed in this chapter with graphical and tabular representations, and a detailed analysis is provided. The results are considered only for few metrics that are useful and reasonable compared to the remaining which are considered in the beginning.

- The Figure 5-1, shows the CPU User, CPU privileged, and CPU stacked for Leveled compaction, date tiered compaction, and size tiered compaction strategies, for a write heavy workload with respect to time in seconds. The graphs show the performances of the system, during the stress. We can observe that the performance of the system is similar for all the compaction strategies. The CPU User percentage and CPU Privileged percentage are as high as around 60 percent and 75 percent respectively, because, the stress is conducted for 1Million operations and 8 threads. CPU Idle reached 100 percent because the CPU idle considers all the operations in the system.

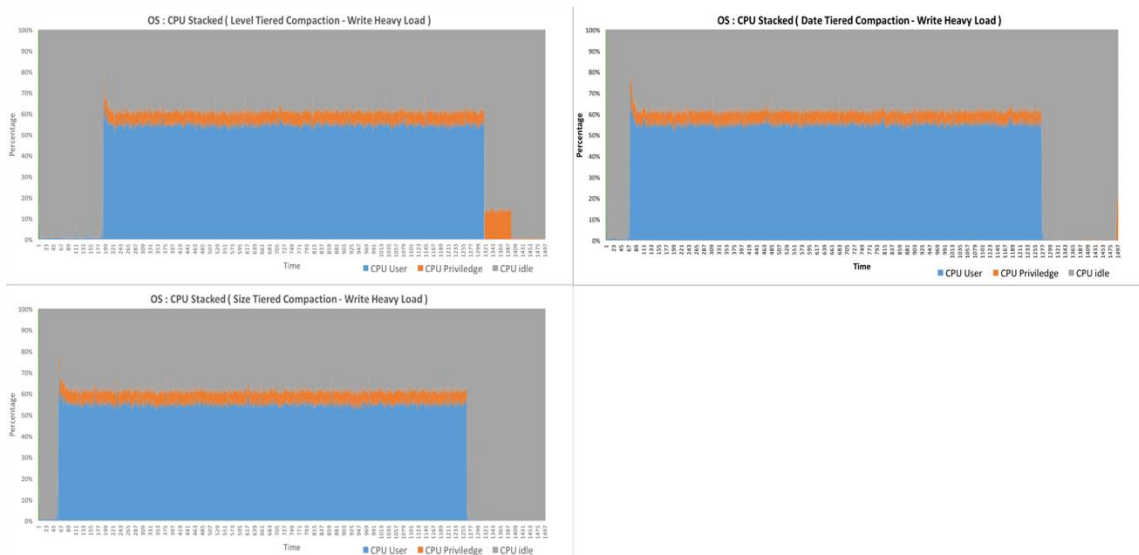


Figure 5-1: CPU User, CPU Privileged, CPU Idle for LCS, DTCS and STCS.

- In the Figure 5-2, the three graphs for Leveled Compaction Strategy, Date Tiered Compaction Strategy, and Size Tiered Compaction Strategy, represents the total number of overall operations with respect to time in seconds, conducted in 20-minute time period. The operations include both read and write. The graphs have many ups and downs, and this is because, for a write operation, cassandra writes directly to the Memtables, hence the write operations are faster, and for read operations, cassandra checks the memtables, as well as the SSTables, hence the operations drop for reads

If we compare the graphs, and also the values in Table 1, we can say that all the three compaction strategies have done similar for write heavy workload.

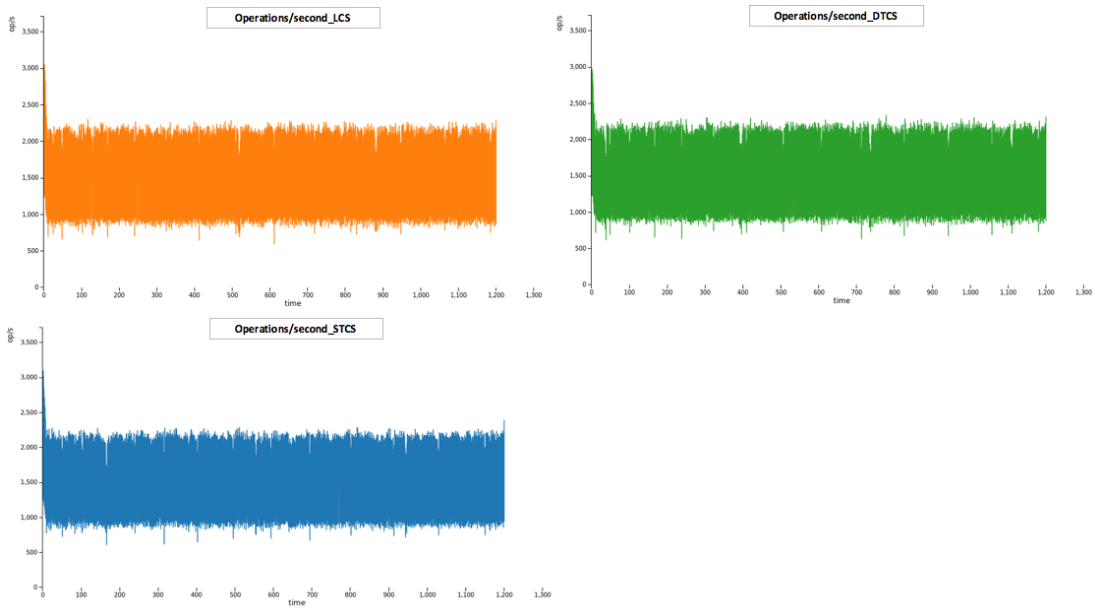


Figure 5-2: Operations/second for LCS, DTCS and STCS.

Compaction	Operations/second
Leveled Compaction Strategy	2350
Date Tiered Compaction Strategy	2355
Size Tiered Compaction Strategy	2358

Table 1: Operations/second.

- The 'latency mean' defines the total mean time taken by a data packet to transfer and to reach back to its source. Figure 5-3 is the graphical representation of latency mean with respect to time in seconds, for write heavy workload. The maximum latency value for all the compaction strategies is '1.5 seconds'. From the graph, we can see that the leveled compaction has an overall constant minimum value of '1 second' and maximum value of '1.5 seconds' for the latency mean. For date tiered compaction, the maximum value remains constant overall, and the minimum value increases from '1 second' with the increase in time, as the data is not compacted with the increase in time, and making the latency increase. For size tiered compaction strategy, the maximum is constant, and the minimum latency value maintain a pattern, with increase and decrease in latency. This is because, when the data is compacted, the latency decreases, and when the data is not compacted, the latency increases.

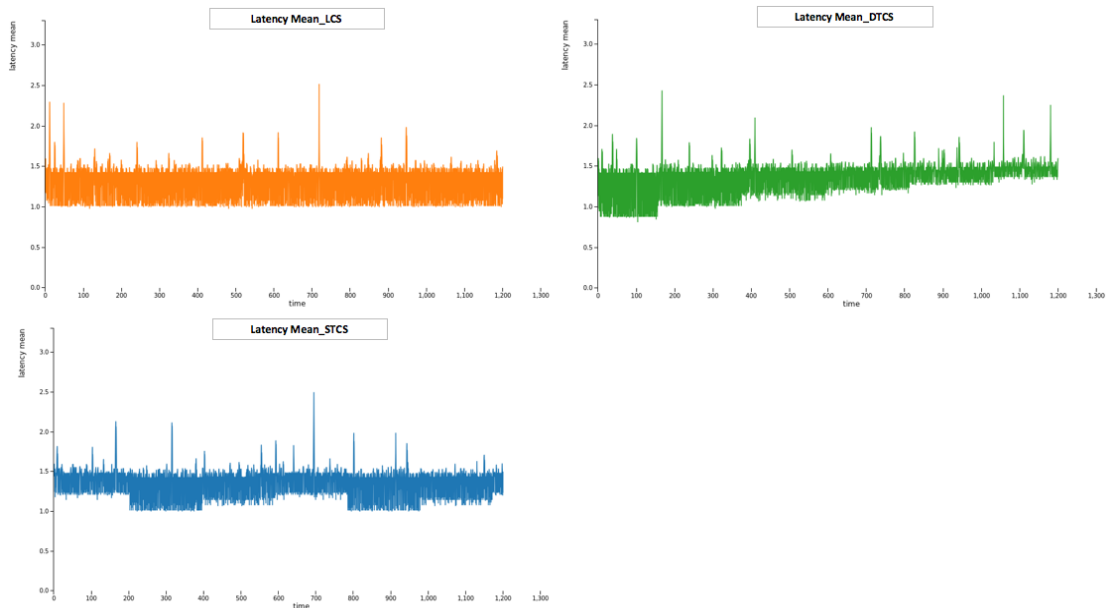


Figure 5-3: Latency Mean for LCS, DTCS, and STCS.

- In Figure 5-4, we can see the total number of compactions with respect to time in seconds, for all the compaction strategies in the entire write heavy process. The level tiered compaction has maximum compactions of 19, as the data is compacted for every level of data, and the overlapped SSTables are re-compacted. For date tiered compaction, the compaction is done base on the time frame, it has less number of compactions in 20-minute time frame. In Size tiered compaction, as the data is compacted based on the size of the SSTable, there are less number of SSTables formed in the 20-minute stress and hence there are less number of compactions compared to leveled compaction strategy. Size tiered compaction, and date tiered compaction has almost equal number of compactions.

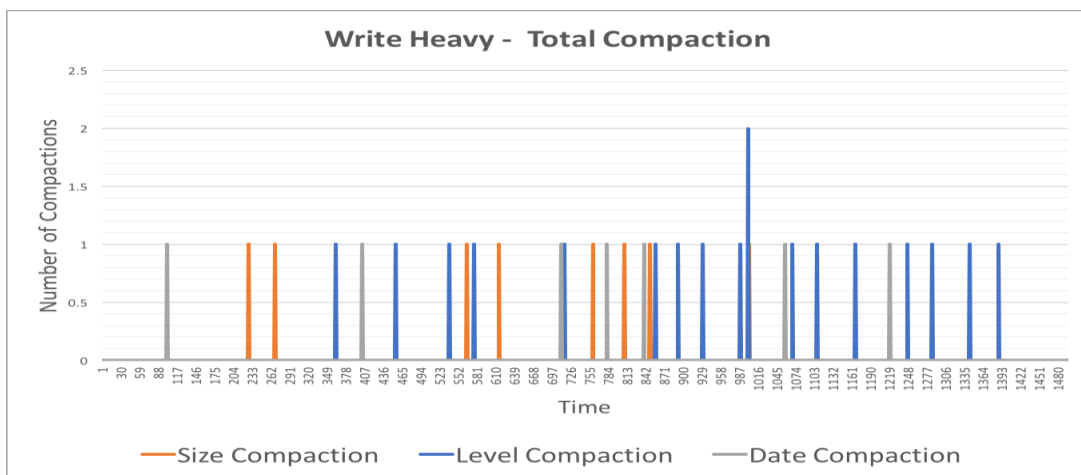


Figure 5-4: Total Compactions for LCS, DTCS, and STCS.

Compaction	Total Compactions
Leveled Compaction Strategy	19
Date Tiered Compaction Strategy	7
Size Tiered Compaction Strategy	8

Table 2: Total Compactions.

- As there are a quite large number of operations for 1 Million operations and 8 threads, the disk operations and disk utilization grow over time. In Figure 5-5, the disk usage percent of all the compaction strategies with respect to time in seconds can be observed. From the graph, we can observe that, size tiered compaction and date tiered compaction, utilizes the disk in similar way. For the leveled compaction, the disk usage grows over time, as the size of the SSTables on the disk is increased over time for increased number of levels. If the SSTable in the first level has ‘x’ amount of data, then the size of the data in its next level is ‘x×10’. For leveled compaction, at time around ‘1001 seconds’, there is a drop from ‘9.7’ to ‘8.8’. This is due to the heavy compaction done for large number of bytes. This can be verified from the Figure 5-4, and Figure 5-7.

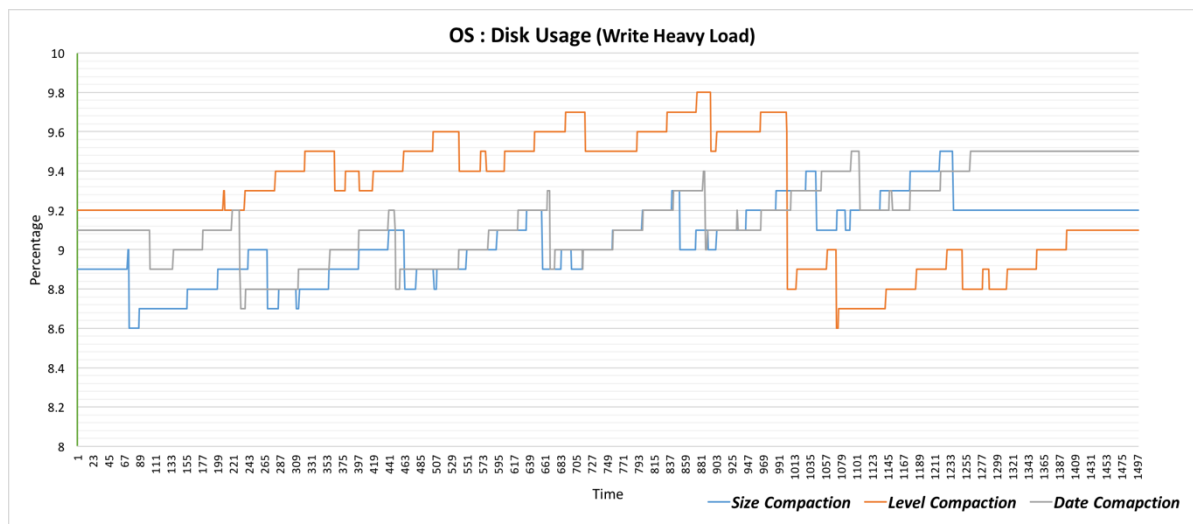


Figure 5-5: Disk Usage Percent for LCS, DTCS, and STCS

- In Figure 5-6, number of disk reads with respect to time in seconds can be observed. From the graph and from the Table 3, it can be observed that, for size tiered compaction strategy, there are three reads. This is common, as the workload is write heavy, and there are only 10 percent of reads in the total data generated. For date tiered compaction strategy, there is only one read. The data generated for date tiered compaction is according to the time, and the workload is read heavy, hence there is only one read for 20-minute time frame. There are no reads for leveled compaction, because, for leveled compaction the data generated will most likely will be in the memory itself or basically within the same SSTable. Since the reads

generated in the stress are very less, and 90 percent of all the reads will stay in the first level, we cannot find any reads in the graph.

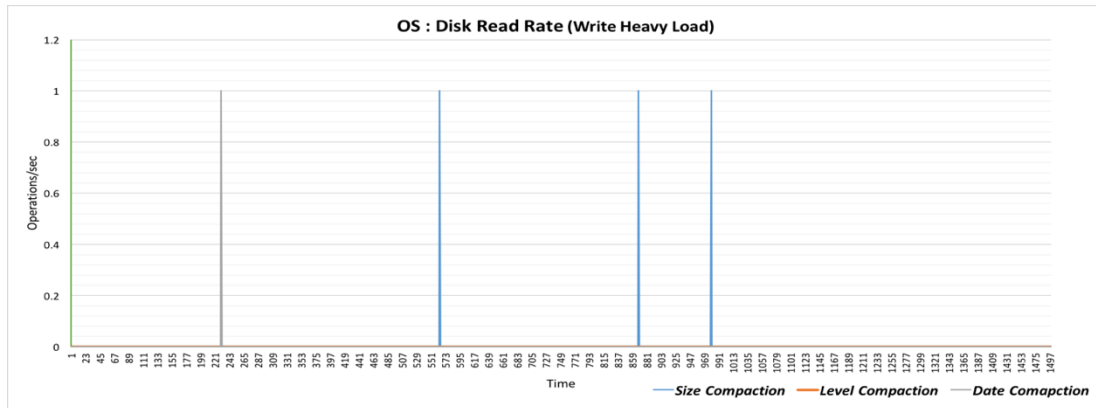


Figure 5-6: Disk Read Rate.for LCS, DTCS, and STCS

Compaction	Reads
Leveled Compaction Strategy	0
Date Tiered Compaction Strategy	1
Size Tiered Compaction Strategy	3

Table 3: Total Disk Reads.

- Figure 5-7 represents the graphs for total bytes compacted with respect to time in seconds for leveled compaction, size tiered compaction, and date tiered compaction strategies. In leveled compaction graph, we can see a huge compaction of 308281344 bytes, which is around 308MB. The size of the level in leveled compaction, increases ten times of the previous levels, and thus the count of total bytes compacted will also increase. Though there are other compactions performed, they cannot be represented in the graph, as they are smaller than this huge compaction. For size tiered compaction, there are two large compactions of 80988 bytes and 80686 bytes, performed at 269 seconds and 849 seconds respectively in 20-minute time frame. These large compactions are because, as the time increases, the size of SSTables increases, and it compact with other SSTable of similar size. The smaller compactions are due to the merging of smaller SSTables, which further form larger SSTables. For date tiered compaction we can observe only one large compaction of 40192 bytes at 102 seconds in 20-minute time frame. This large compaction is also due to the merging of larger SSTables. These large compactions can increase with the increase in time, for date tiered compaction strategy.

The huge compaction in the leveled compaction strategy results in the freeing of the cache memory, which can be seen in the Figure 5-8.

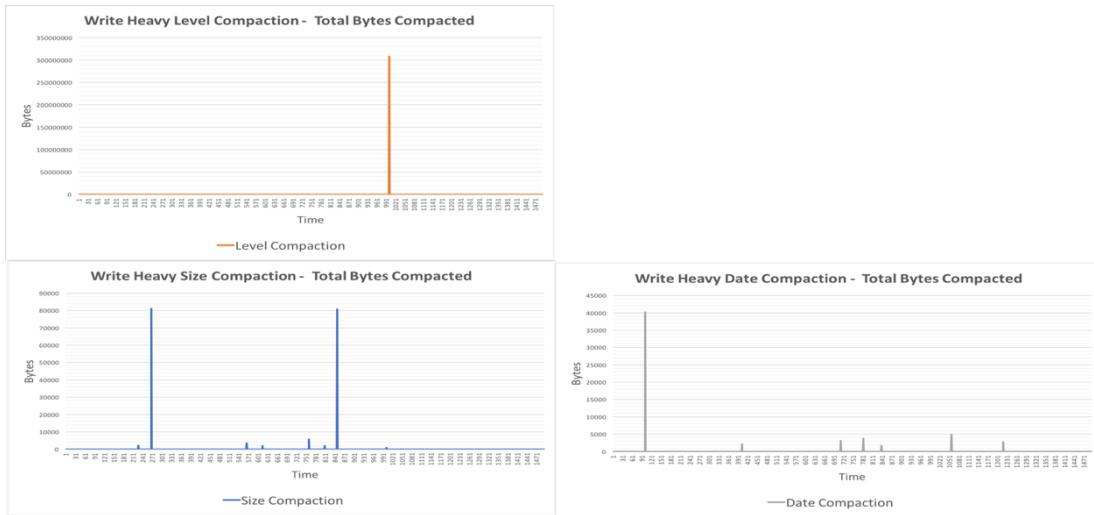


Figure 5-7: Total Bytes Compacted for LCS, DTCS, and STCS.

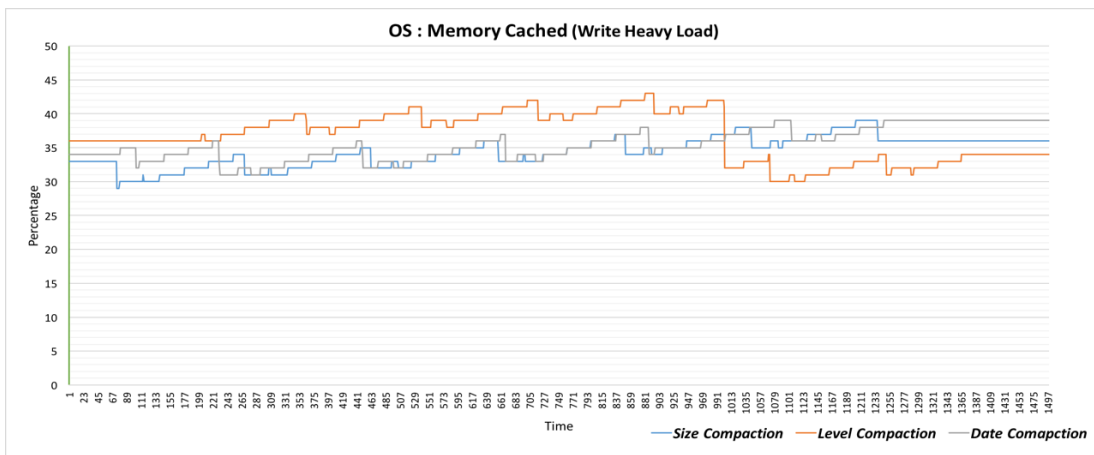


Figure 5-8: Memory Cached for LCS, DTCS, and STCS.

6 CONCLUSION AND FUTURE WORK

The purpose of this thesis was to analyze the performances of compaction strategies in Apache Cassandra. A detailed literature review has been done to understand the working of different compaction strategies in Cassandra. From the literature research, a great knowledge on NoSQL databases, and the difference between Relational Database Management Systems has been obtained. The architecture, data model, detailed technical working of Cassandra database, and the compaction strategies in it are understood by exploring the Datastax documentation.

A decent amount of time was taken to find the appropriate performance metrics for analyzing the success of Size Tiered Compaction Strategy, Leveled Compaction Strategy, and Date Tiered Compaction Strategy in Apache Cassandra. The initially prepared time plan was followed successfully until the literature review, but, the execution of experiment took some extra time, to make the installations, check the compatibility of the versions, and some technical changes that are made to the server provided. As Cassandra is open sourced, there is always an issue either at the starting of the experiments or in the middle of the experimentation, but, it was somehow managed later.

Several meetings are made with the team working on Apache Cassandra in Ericsson, and also with the supervisors at BTH, to make the necessary changes in decisions taken, and the method we followed. The performance metrics that are to be considered for the analysis of the compaction strategies are finalized by taking the advice of supervisors at Blekinge Institute of Technology, and the Cassandra team in Ericsson. Different existing methods were practiced in collection of the performances for the considered metrics, but finally two tools were developed in Jython for compaction metrics and in Python for system or OS metrics, to collect the unbiased instantaneous values of the performances in to a csv file. The graphs were generated in Microsoft Excel by using the data in the csv file. Analysis of the results were made based on the understanding from the literature review, and considering valuable advices from the supervisors of the project and Ericsson team.

From the results and the analysis, it can be concluded that, Leveled Compaction Strategy is a good option for a write heavy (90/10) workload, using default cassandra stress model. Though the CPU User, CPU Privileged, and CPU idle is similar for leveled compaction strategy, Size Tiered Compaction Strategy, and Date Tiered Compaction Strategy during the stress, the conclusion is drawn by considering the compactions and its effect on the system. There are large number of compactions and heavy data compacted by using leveled compaction strategy in 20-minute time frame, compared to size tiered and date tiered compaction strategies. Large number of compactions and huge amount of data compacted is good for any kind of database, as it allows more data in to the database. This is a considerable case for any company that generates large amounts of data in a regular basis. This also improves the system performance by reducing the disk utility, and reducing the cache memory at that instance of time.

Date Tiered Compaction Strategy, and Size Tiered Compaction Strategy performed similar for the stress tests conducted. Date tiered compaction works good for time series data model rather that default cassandra stress model, as the compaction is done according to the time scale. Size Tiered Compaction Strategy compact only small amounts of data in the given time frame, as compared to Leveled Compaction Strategy.

Linking to Research Questions:

- 1. What are the performance metrics to be observed to evaluate the success of a compaction strategy for a write heavy data?**

The performance metrics that are necessary to consider for the evaluation of the performances of compaction strategies in Cassandra are finalised by the understanding from literature research, considering the advices from the supervisors and Cassandra team at Ericsson.

There are many metrics considered for the performances which are stated in the Chapter 4.2, but only few of them are used for the analysis and evaluation. The performance metrics considered involve both system metrics and compaction metrics. They are: CPU User, CPU Privileged, CPU Idle, Disk Usage, Disk Read Rate, Memory Cached, Operations/Second, Latency Mean, Total Number of Compactions, Total Bytes Compacted.

- 2. How to measure the performances of different compaction strategies of Cassandra using default Cassandra stress model for Write heavy (90/10) workload?**

A couple of tools were developed using Jython and Python scripting language to collect the instantaneous values for the considered metrics. The Jython script was used to collect the performances for cassandra metrics. The script connects to the JMX and MBeans server to collect the cassandra metrics. Python script was used to collect the performances for system or OS metrics. The script uses 'psutil' to collect the OS metrics. Both the scripts store the collected values in to different csv files which are later used to generate the graphs in Microsoft Excel.

- 3. Which compaction strategy is appropriate for a write heavy (90/10) workload, using default cassandra stress model?**

Leveled Compaction Strategy is appropriate for a write heavy (90/10) workload, while using default cassandra stress model. The leveled compaction strategy manages to make maximum number of compactions and the compactions are performed for larger amounts of data in a small time frame, as compared to size tiered compaction strategy, and date tiered compaction strategy. The heavy compactions result in dropping of the disk utility and cache memory at that instance of time.

Future Work:

As the solutions for efficient database usage is increasing with the evolution of technology, there is always a need to explore new ways to find an effective solution. This thesis can further be prolonged to test for different stress techniques, and different data models in Cassandra. This project can be repeated for a multi node Cassandra cluster, with a large environment. The tools developed for collecting the OS metrics in this thesis work can be used for various experimentations in any field. The tool developed for collecting Cassandra metrics can be used for any projects on Apache Cassandra.

REFERENCES

- [1] P. Bagade, A. Chandra, and A. B. Dhende, "Designing performance monitoring tool for NoSQL Cassandra distributed database," in *2012 International Conference on Education and e-Learning Innovations (ICEELI)*, 2012, pp. 1–5.
- [2] C. Y. Kan, "Cassandra Query Language," in *Cassandra Data Modeling and Analysis*, Packt Publishing, 2014.
- [3] C. A. Frangos, "Using Strategy Maps to Prioritize Learning and Performance Improvement Agendas," *Perform. Improv.*, vol. 46, no. 2, pp. 26–29, Feb. 2007.
- [4] B. R. Chang, H.-F. Tsai, Y.-A. Wang, and C.-F. Kuo, "Intelligent Adaptation to in-Cloud NoSQL Database Remote Backup Between Data Centers," in *Proceedings of the ASE BigData & SocialInformatics 2015*, New York, NY, USA, 2015, pp. 4:1–4:6.
- [5] M. Brown, "Preface," in *Learning Apache Cassandra*, Packt Publishing, 2015.
- [6] N. Padalia, *Apache Cassandra Essentials*. Packt Publishing, 2015.
- [7] V. Johansen, *Object serialization vs relational data modelling in Apache Cassandra: a performance evaluation*. 2015.
- [8] S. Abbireddy, *A Model for Capacity Planning in Cassandra : Case Study on Ericsson's Voucher System*. 2015.
- [9] C. Y. Kan, *Cassandra Data Modeling and Analysis*. Packt Publishing, 2014.
- [10] M. Nicola and M. Jarke, "Performance modeling of distributed and replicated databases," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 4, pp. 645–672, Jul. 2000.
- [11] "WritePathForUsers - Cassandra Wiki." [Online]. Available: <https://wiki.apache.org/cassandra/WritePathForUsers>. [Accessed: 11-May-2016].
- [12] "The cassandra.yaml configuration file." [Online]. Available: https://docs.datastax.com/en/cassandra/2.0/cassandra/configuration/configCassandra_yaml_l_r.html#reference_ds_qfg_n1r_1k_key_cache_size_in_mb. [Accessed: 11-May-2016].
- [13] A. Rafique, *Evaluating NOSQL Technologies for Historical Financial Data*. 2013.
- [14] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, "Performance Evaluation of a MongoDB and Hadoop Platform for Scientific Data Analysis," in *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, New York, NY, USA, 2013, pp. 13–20.
- [15] N. Neeraj, *Mastering Apache Cassandra*. Packt Publishing Ltd, 2013.
- [16] P. N. Shankaranarayanan, A. Sivakumar, S. Rao, and M. Tawarmalani, "Performance Sensitive Replication in Geo-distributed Cloud Datastores," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2014, pp. 240–251.
- [17] "Big Data Platforms: An Overview," 19:00:16 UTC.
- [18] N. S. Kuruganti, *Distributed databases for Multi Mediation : Scalability, Availability & Performance*. 2015.