Thesis no: MSEE-2016:31

Compaction strategies in Apache Cassandra

Analysis of Default Cassandra stress model

Venkata Satya Sita J S Ravu

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

Contact Information:
Author(s):
Venkata Satya Sita J S Ravu
E-mail: vera15@student.bth.se

External advisor:
Jim Hakansson
Ericsson
Karlskrona

University advisor:
Emiliano Casalicchio
Senior Lecturer in Computer Science
Computer Science Engineering
Blekinge Institute of Technology

Faculty of Computing	Internet	: www.bth.se
Blekinge Institute of Technology	Phone	: +46 455 38 50 00
SE-371 79 Karlskrona, Sweden	Fax	: +46 455 38 50 57

ABSTRACT

Context. The present trend in a large variety of applications are ranging from the web and social networking to telecommunications, is to gather and process very large and fast growing amounts of information leading to a common set of problems known collectively as “Big Data”.

The ability to process large scale data analytics over large number of data sets in the last decade proved to be a competitive advantage in a wide range of industries like retail, telecom and defense etc. In response to this trend, the research community and the IT industry have proposed a number of platforms to facilitate large scale data analytics. Such platforms include a new class of databases, often refer to as NoSQL data stores. Apache Cassandra is a type of NoSQL data store. This research is focused on analyzing the performance of different compaction strategies in different use cases for default Cassandra stress model.

Objectives. The performance of compaction strategies are observed in various scenarios on the basis of three use cases, Write heavy- 90/10, Read heavy- 10/90 and Balanced- 50/50. For a default Cassandra stress model, so as to finally provide the necessary events and specifications that suggest when to switch from one compaction strategy to another.

Methods. Cassandra single node network is deployed on a web server and its behavior of read and write performance with different compaction strategies is studied with read heavy, write heavy and balanced workloads. Its performance metrics are collected and analyzed.

Results. Performance metrics of different compaction strategies are evaluated and analyzed.

Conclusions. With a detailed analysis and logical comparison, we finally conclude that Level Tiered Compaction Strategy performs better for a read heavy (10/90) workload while using default Cassandra stress model , as compared to size tiered compaction and date tiered compaction strategies. And for Balanced Date tiered compaction strategy performs better than size tiered compaction strategy and sate tiered compaction strategy.

Keywords: Big data platforms, Cassandra, NoSQL database.

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 Srinand Kona for his continuous support and partnering for this Master thesis project.

CONTENTS

LIST OF FIGURES

ABBREVIATIONS

CPU	**Central Processing Unit**
CQL	**Cassandra Query Language**
DTCS	**Date Tiered Compaction Strategy**
JMX	**Java Management Extension**
LCS	**Leveled Compaction Strategy**
NOSQL	**Not Only Structured Query Language**
STCS	**Size Tiered Compaction Strategy**
SQL	**Structured Query Language**
VM	**Virtual Machine**

1 INTRODUCTION

This thesis is based on the fact that Big data on cloud based database is booming over time. The number of user on Drop Box alone has increased from 1 million to 5 million over five years. Data on cloud drives is storming out every day. To handle these heavy concurrent operations, a powerful database management system is required. Facebook came up with Cassandra as an initial solution to handle heavy read and write operation on messenge[1]r. This showed greater performance benchmarks than previous database management systems. This motivated many other large scale companies to implement Cassandra.

Apache Cassandra is a open source distributed database management system that is designed for storing and managing large amounts of data across commodity servers. (DDS)[2]. Cassandra provides fault tolerant high performance, high scalability.

Data was partitioned among all nodes of the cluster. Each node communicates with each other through the Gossip protocol, which exchanges information among the cluster every second. Cassandra is advantageous than many other data bases, No single point of failure, No need for separate caching layer, No need of special hardware or software, Data can be compressed and understands CQL language[3]. Can replicate data among different physical data center racks.

Many companies have successfully deployed and benefited from Apache Cassandra including some large companies such as Apple, Netflix, Instagram, Spotify, Ebay, Rackspace[4]. In Cassandra all nodes play an identical role, there is no concept of master node. It is capable of handling large amounts of data as easy as it can manage much smaller amount of data and user traffic.

The Apache Cassandra data model allows for new entities or attributes to be added overtime. Cassandra supports an incredible array of language drivers to ensure that your application runs optimally on Cassandra. Cassandra is essentially a hybrid between a key-value and column oriented data base management system. Cassandra cannot do joins or sub queries, Rather Cassandra emphasizes renormalization through features like collection[5].

Cassandra is a Java based system that can be managed and monitored via Java Management Extensions (JMX)[2]. There is no single point failure. Data is distributed across the cluster. But there is no master every node can service any cluster. Read and write throughput both increases linearly as new machines are added, with no downtime or interruptions to applications.

Cassandra introduces CQL language, a SQL alternative to the traditional RPC interface. CQL is a simple API meant for accessing Cassandra.

Apache Cassandra is considered over HadoopDB and MongoDB in this thesis because it is constantly outperforming and widely used cloud based data management system[5]. Cassandra has a master less ring architecture which enables fault tolerance and has capability to handle high loads. Apache Cassandra is highly scalable which gives the user great scope of tuning the database according to load and improve performance. Cassandra database nodes can be geographically distributed; this gives a great advantage in handling big data around the world[6].

1.1 Thesis statement

This thesis discusses about the selection of compaction strategies in Cassandra stress model. To benchmark Cassandra performance metrics Cassandra provides a developer tool which can generate random read and write requests according to developers choice using Cassandra stress tool a developer can estimate how Cassandra behaves in real time with proposed configurations. The performance of compaction strategies are observed in various scenarios on the basis of three use cases, Write heavy – 90/10, Read heavy – 10/90 and Balanced – 50/50, for a default Cassandra stress model. Considering Write heavy, Read heavy and Balanced providing proper selection of compaction strategies suggests when to switch from one compaction strategy to other.

1.2 Objectives

1. To investigate on how the compaction strategies works.
2. To investigate how date tiered, size tiered, levelled compaction strategy works and also their behaviour in Write heavy (90/10), Read heavy (10/90), and Balanced (50/50) use cases in a default Cassandra stress model.
3. To find appropriate performance metrics to find the effect of the compaction strategies on the system.
4. To conduct measurements on the field to compare the performance of the compaction strategies.

1.3 Research questions

1. How to measure the performance of a compaction strategy in various scenarios?
   - The compaction strategy is configured on a node and read heavy and balanced stress test is done, during the test the system performance metrics like CPU utilization, Disk throughput, memory cached and other performance metrics are monitored similarly Cassandra metrics like number of compactions total bytes compacted and other performance metrics are monitored with the data collected fro monitoring best compaction strategy is derived.

2. How to measure the performances of different compaction strategies of Cassandra for default Cassandra stress model in Read heavy (10/90), and Balanced (50/50) use cases?
   - A couple of tools were developed using Jython and Python scripting language to collect the instantaneous values for the considered metrics.

3. Which compaction strategy is appropriate for Read heavy (10/90), and Balanced (50/50) workload using default Cassandra stress model?
   - Date tiered compaction strategy is appropriate for Read heavy (10/90), and for Balanced(50/50) as well.

1.4 Split work

In this section, we write the work shared among us. Which parts contributes to which person.

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, Swaroopa Ravu and, Srinand Kona, 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 involve 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 Swaroopa, 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 Swaroopa. The two different tools that are used for collecting the performances during the experiments is developed by Srinand.

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

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

2 PRELIMINARY DISCUSSIONS

2.1 Big Data Systems

Traditional MySql database systems cannot handle large amount of data uploaded and managed on cloud based storage drives, this data is called big data. There are some powerful databases to manage big data and they are called big data serving systems. Some of the most popular big data systems are Hadoop MongoDB Hbase and Cassandra. Most of these big data management systems use NOSQL database[7]. Unlike rest of the big data systems Cassandra offer automatic data distribution and customizable replication.

2.2 Cassandra

Cassandra is an open source NOSQL database system. This gives Cassandra massive scalability and architecture of distributed database system. As it is a JAVA based software, It gives tunable consistency and better monitoring applications. Cassandra has a master less ring architecture in which all the nodes are connected using peer to peer connection[8]. To explain the data structure of Cassandra here are some key concepts to be acknowledged. All the data in Cassandra is replicated if replication factor is greater than one and stored in the nodes. These nodes can be geographically distributed but must stay under single cluster. The cluster holds a number of data centers which can inter communicate in network topology. Database holds nodes. These nodes can only communicate with the nodes of other datacenters only if local data center allows it. Most commonly used topology is simple topology in this topology we have one data center and multiple nodes are installed in that data center. This data center holds data of entire data base.

Another key factor to be learned before understanding Cassandra is replication factor. To avoid Cassandra loosing data even one of the nodes is down replicating data among the nodes is used as fault tolerance mechanism. Replication factor defines how data should be distributed among the nodes[9]. For example, in a three node network if a key space is created with replication factor one. If data is inserted into the key space the data is divided into three parts and each part is stored in each node. In same way the replication factor is given as 3 entire data is stored on every node. Data can be inserted or managed among the node using command line interface or JQuery method.

2.3 Cassandra Data Model

Cassandra data is stored on per column basis. Every column stored has its key store value. The key store value holds the information about location of data and its time stamps. Whenever the column is compacted its information is purged from the commit log so it becomes difficult for the Cassandra to find the data if the key stores values are not organized properly or went missing[10]. Changing compaction strategy will help organizing key store value. Organizing the key store value will finally impact on read performance by decreasing the read path[11].

The write throughput of the disk depends on several major factors a couple of them are total bytes compacted and number of compactions. Total bytes compacted depends on column stored in SS table. Changing in compaction strategies will change the method used to merge SS table this will result in either increase or decrease on number of compactions. Increasing in number of bytes compacted will allow Cassandra to write more bytes into the disk[12]. Cassandra doesn't offer many configurations to improve write performance but we can configure Cassandra using compaction strategies to decrease disk space used to store data. The outermost container is known as the cluster. For failure handling, every node contains a replica, and in case of failure the replica takes charge[13].

2.4 Compaction in Cassandra

To understand compaction, knowing the write path of Cassandra is important when a data is inserted into Cassandra data base the data is stored in the form of columns these columns are first written in a commit log and memtable. Memtable cannot store the entire database so when the threshold value is reached memtable flushes the data SS tables. After reaching a threshold time the columns stored in SS tables are merged together and stored in the disk the process of merging the columns together in SS tables is called compaction there are three different compactions Cassandra offers. Each compaction has its own benefits and disadvantages which will be further discussed in this thesis.

Compactions plays a major role in data management, write and read performance[11]. To define the affect of compaction strategies on write performance. A detailed write path is explained further. To read old data in Cassandra data base the merged tables data location and time stamp are saved in memtable disk Cassandra after checking rowcache memory and keycache memory for disk location[15]. The similar these time stamps are arranged the quicker Cassandra reads the data. The compaction process in Cassandra merges keys, combines columns, evicts tombstones, consolidates SSTables, and creates a new index in the merged SSTable. Compaction is implemented by a compaction strategy and Cassandra made available three of them: SizeTieredCompactionStrategy (STCS), DateTieredCompactionStrategy (DTCS) and LeveledCompactionStrategy (LCS).

2.4.1 Size tiered compaction strategy (STCS)

Size tiered compaction strategy is a default compaction strategy. This compaction strategy set-off a compaction when multiple SS tables of same size are present means when the disk has specific number of SS tables with similar size. These compacted tables can be compacted further with the same size of tables to form larger SS tables,this process is continuous until the compaction ends. This compaction strategy is good for insert-heavy and general workloads. This compaction works better when the data is once generated and not modified later. It combines multiple SS tables which belong to a particular data size range. Size tiered compaction strategy doesn't give any guarantee about column distribution of particular row it may happen that columns of particular row-key belong to different SS tables.

2.4.2 Leveled compaction strategy (LCS)

Leveled compaction strategy is best read-heavy workloads. This strategy groups SS tables into levels each will have fixed size limit which is 10 times larger than the previous level. In Leveled compaction strategy the SS tables are fixed and relatively small in size divided into different levels. . In the first level, level 1, the new SSTables are created in which the flushed data from memtables is present. In the levels other than level 1, they are exponentially increasing in size. For example if level 1 is filled, any new SS tables being added to that level are compacted together with the existing table that contains overlapping data. The excess tables in level 1 will over flow to level 2. The data in these SSTables is sorted by Log-structured merge-tree data structure. The SS tables are guaranteed with the non overlapping with each level. If any data is overlapped when it is promoting to the another level overlapped tables are re-compacted.

2.4.3 Date tired compaction strategy (DTCS)

Date tiered compaction strategy is a newest compaction strategy designed until now it is designed for use with time series data, but it can also used for other kinds of data. Date tiered compaction strategy stores data written within the same time period in the same SS table. Data is inserted in a organized way according to the timestamp. The basic idea is to act similarly to size tired compaction strategy, but instead of compacting based on SS table size compact based on SS table age. DTCS helps a lot in purging tombstones, because when they reach the oldest SS table, Cassandra simply removes any tombstone that is older than GC grace period. An SS table that is older than all have no reason to hold tombstones beyond the grace period. The separation of old and new data is good for time series. It also holds an advantage against the other strategies at purging deleted data quickly and predictably.

2.5 Cassandra Stress Tool

To benchmark Cassandra performance metrics Cassandra provides a developer tool which can generate random read and write requests according to developers choice using Cassandra stress tool a developer can estimate how Cassandra behaves in real time with proposed configurations. Cassandra stress tool runs in deamon this helps the developer to configure Cassandra nodes on run time. This can lead to propose a dynamic adaptation strategy. Other tools like YCSB[14] cannot provide this feature. Cassandra stress tool provides three workload stress tests. In read heavy workload user can either provide number of read requests or the duration read requests. He can also mention the key space and the node to be tested other additional of tests are sending uniform requests and defining replication factor etc., the similar features are provided for read heavy and balanced workloads.

3 RELATED WORK

This thesis discusses about the selection of compaction strategies in default Cassandra stress model. The performance of compaction strategies are observed in various scenarios on the basis of three use cases. The thesis explains the concepts of Cassandra which is a NoSQL (Not only Structured Query Language) database. It also discuss about Cassandra capacity planning.

The paper [2] "Evaluating NOSQL Technologies for Historical Financial data" gives a good understanding to find an efficient solution for storing and processing the huge volume of data for certain variants. The author discusses about choosing a reliable, distributed, and efficient NOSQL database at Cinnober Financial Technology AB. The main focus is to find out which database is the preferred choice for different variants. In this regard, the performance test framework for a selected set of candidates has also been taken into consideration. The main disadvantage is that the author only compared the RDBMS to the NOSQL database. Advantage is he gave an attempt to contribute the research in the field of NOSQL databases which compares one such NOSQL database Apache Cassandra with Apache Lucene and the traditional relational database MYSQL for financial.

The paper [5] presents an overview on the comparison of non-functional properties for different types of NoSQL DBMS. The most used DBMS are MongoDB, Redis and Apache Cassandra. After working with the databases and performing YCSB Benchmarking the conclusion is that is that if the database should handle an enormous amount of data, Cassandra is most probably best choice. The paper concluded that if the database needs to be flexible and versatile, MongoDB is probably the best choice.

The paper [10] discusses about the issues related to scalability challenges due to I/O bottle necks faced by large scale cloud based services with database performance as the major concern. The performance of Object serialization approach which is a type of Data Model Optimization has been tested using the Apache Avro library in the Cassandra database. Two serialized data models were compared a traditional database model. The performance was found to be improved by Avro Serialization. Serialization granularity which is defined by a serialization model was found out to affect the performance benefit to certain extent. It was seen that serialization by itself will not perform better than relational modeling in all use cases and data model optimization should be given more priority to improve database throughput in Cassandra through serializations.

The paper [16] was aimed at the reducing the Tail latency for reads in the Apache Cassandra Database system by implementing the new replica system algorithm called C3. The authors found out that C3 decreases the Tail latency for the read in the Apache Cassandra.C3 was found to have shown no improvement in terms of production load.C3 was also implemented on the client side in the Data Stax Java driver to remove caveat of token aware clients. Though client side implementation gave positive results, benchmark results showed lot of variance. Hence it was concluded that C3 works efficiently for systems with homogenous row sizes where clients are not token aware.

Modern computer systems enables to extract valuable information such as customer behavior, trends and more, from the large amounts of raw data. The author deals with Cassandra which handles with large amount of highly available data. Cassandra capacity planning i.e Cassandra data storage dimensioning predicts the amount of disk storage required in the case of a particular product being deployed by it. This study identifies the factors affecting the disk space in Cassandra data storage system. This capacity planning is based on whether it is Cassandra specific and Product specific. This makes the capacity planning process more efficient. And these factors leads to develop a model which predicts the disk storage for Ericsson's voucher system. [17]

[18] In this research it is analyzed that distributed databases in terms of performance, scalability and availability. The inter-relations between performance, scalability and availability of distributed databases are also analyzed. The distributed databases that are analyzed are MySQL Cluster 7.4.4 and Apache Cassandra 2.0.13. Performance, scalability and availability are quantified, measurements are made in the context of Multi Mediation system. In Cassandra there are good opportunities for scalability and it has, well suited for intended type of data. And other model, Hadoops distributed computing capabilities widely might meet our wishes. So, the combination of these two software's provides a various range of functionalities. Real-time and larger analysis calculations are considered. Cassandra's high reliability and fast write performance makes it suitable for storage of streaming log data.

The both qualitative and quantitative methods are used to carry out this research. For evaluation, qualitative study is made for databases selection. To evaluate quantitatively the performance, a benchmarking harness application is designed for distributed database in the context of Multi Mediation. Several experiments are designed and performed using the benchmarking harness on the database cluster. For a good user experience, there is need to keep response time low.

[19] The average INSERT response time and throughput results supports Apache Cassandra low availability configuration. On comparing, MySQL Cluster average and Apache Cassandra the SELECT response time is better for Cassandra considering more number of client threads, both in high availability and low availability configurations. Although Apache Cassandra outperforms MySQL Cluster, the support for transaction and ACID compliance are not to be forgotten for the selection of database. Other parameters to be considered for selection of database within an organization organizational choices, resource utilizations, costs for developing etc. for distributed databases, further evaluation is opted.

By testing extensively using various stress tools, we can find that C3 indeed decreases the tail latencies of Cassandra on generated load. Also, by evaluating C3 on production load, we cannot show any particular improvement. And this might be mostly due to the variable size records in the data set. Also might be due to token awareness in the production client. A client-side implementation of C3 in the DataStax Java driver is also used to remove the caveat of token aware clients. Considering the client-side implementation , it gives positive response but as lot of varience observed in benchmark results we believ the results to be too inconclusive to confirm that the implementation works as intended. We conclude that the server-side C3 algorithm will work effectively for systems with homogeneous row sizes where the clients are not token aware.

4 TEST ENVIRONMENT

A BTH server is allotted for conducting thesis experiments this server can be accessed using internet gateway. A profile is created on this web server which can be accessed using a password generated during initialization of the profile. Initializing profile makes node secure from other traffic in the web server. This makes the cluster isolated and safe from the attackers (hackers). In this virtual machine Cassandra is installed and configured to create a single node cluster in the web server. A Cassandra node is installed on BTH cluster and accessed on a remote host. To completely make the environment cloud based the installed node is configured and tested over stand alone SSH client. It is a single node network so data is not replicated.

4.1 Web Architecture

A web server pre-installed in BTH is granted access to operate on a remote host on this web server a virtual machine is installed with required specifications and isolated network. As this VM is installed on application level it is easily scalable and can be used like a real time Cassandra node. To make this VM Cassandra node Cassandra open source software is downloaded and installed on this virtual node. This web server is granted access to other students who are trying to conduct similar experiments. To avoid other students interference in our network the virtual machine is protected with a password. The virtual machine network can only accessed through an SSH client over a remote host only if the host have authentication password[20].

4.2 Cassandra Architecture

We deploy a single node Cassandra cluster on the web server and configured as it is a node Cassandra data base we need not worry about inter node gossip. A key space is created using command line interface. The replication factor is set to 1 and topology is set to simple topology. All the configurations in Cassandra are set to default so we have only one seed and one client in the cluster. After installation the firewall of Cassandra ports are disabled, so that it can hear client request and connective JMX memory. Cassandra tool architecture comes with a built in tool called Cassandra metrics which store all its data in JMX memory using 7199 port. The developed Jython tool collects the Cassandra performance metrics values JMX port using the URL http.//localhost;7199/ the metrics to be collected are defined in .yaml file which is only readable by the code[2].

4.3 Monitoring Tool Architecture

To monitor Cassandra node in a real time datastax, opcenter is installed in the virtual machine at the first Datastax opcenter provides real time graphs which helps to understand Cassandra behavior at different workloads and configuration this helps to reduce time wasted on unwanted test runs. Datastax opcenter retrieves information from JMX port and sense them to https deamon listening port this deamon listening port creates an web URL. This web URL can be accessed over web browser. In the web browser a web GUI is displayed with Cassandra performance metrics graph. Later

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

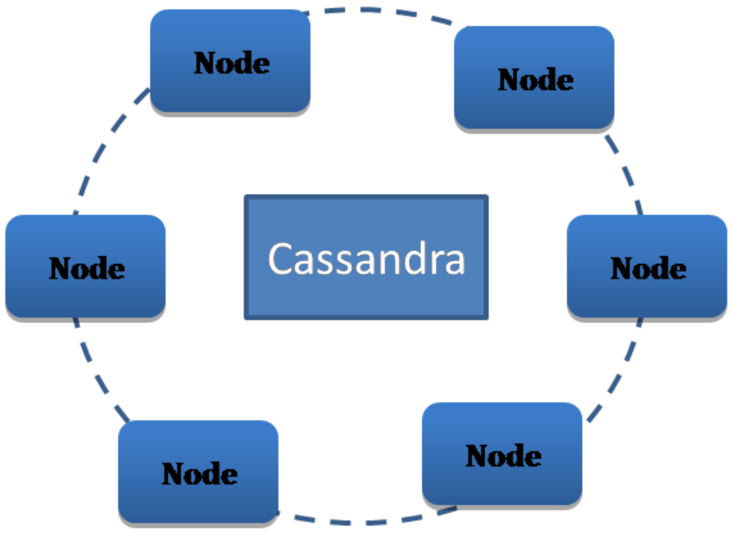

Figure 1 Cassandra cluster ring architecture

5 METHEDOLOGY

5.1 Choosing System Specifications

Before starting the experiment the stable versions of software used in this thesis are tested and verified before implementing. Hardware specifications are also estimated. The stable version of virtual machine is installed in the remote server and configured. The latest table version of Cassandra 3.5 is installed in the node. Stable version of Cassandra 3.5 is installed in the virtual machine. As the virtual machine specifications are scalable. The software defined specifications are configured to run Cassandra without lag. This way the better specifications for virtual machine and stable version of Cassandra are installed and tested. Now the monitoring tools are selected and tested in similar manner. The functionalities like node tool utility, command line interface and Cassandra stress tool are understood.

5.2 Deploying a Single Node Cassandra Data base on Remote Server

After Cassandra is installed in remote server Cassandra must be configured to create a cluster, data center and a node that is fully functional. A configuration file can be used to change Cassandra listening and seed IP addresses in our case the listening IP address and seed IP address are set to local host to create a single node cluster Cassandra stress tool generates read and write request randomly using random variables so the same read requests or the write requests are not generates twice.

5.3 Stress test

The Cassandra data base is stressed with write heavy read heavy and balanced workloads and metrics are monitored and logged. Cassandra stress tool is used to send multiple read and write requests to the node. Cassandra stress tool also comes with the options like choosing uniform distribution for read and write request specifying the key space to be tested and mentioning compaction strategies etc.

The logged data is useful to assist thesis with further process. The rate of requests generated by Cassandra stress tool is understood and controlling rate of requests behavior is mastered this helps to simplify thesis and save time on unnecessary tests.

The performance metrics that could be affected are added in the monitoring tool to log the data. After defining the performance metrics and workload scenarios we are ready for our first test run.

The first compaction strategy we choose to test is Size tiered compaction strategy (STCS). In Cassandra stress tool to create write heavy workload 9:1 ratio is given. This test is run for 30 minutes which can be mentioned in Cassandra stress command. The system metrics like disk write throughput read throughput and disk utilization and Cassandra metrics like number of compactions total compacted bytes memtable blocked operation throughput are monitored and the result is saved for further analysis this process is repeated with read heavy. In read heavy 1:9 write to read ratio is given on Cassandra stress tool.

Cassandra three compaction strategies are saved in each .YAML file which is duplicated from Cassandra.yaml file assigning this while running Cassandra stress tool will reduce the run time. Every compaction must be tested with three different workloads. The performance metrics are collected for every stress test.

5.4 Collecting Metrics

These performance metrics are collected from JMX port by tool developed in Jython. The tool gives output of graphs and a log file. The metrics are also continuously monitored on datastax opcenter to understand the behavior of compaction strategies at different level of stress test[21].

These data graphs are generated in time series so for deeper study on Cassandra performace behavior with change in compaction strategies the log files are also analyzed and relative graphs are processed.

5.5 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.

5.6 Data Analysis

Just by looking at time series graphs and log files it is difficult to tell about change in performance metrics with respective to other basic performance metrics like write requests per second and number of requests etc. to compare between any two performance metrics the logged data is processed and a graph is created against those two performance metric values.

These graphs provide sufficient knowledge to decide between compaction strategies at different workloads. By end of the entire experiment, the best compaction strategy at every instant of work load is derived and result is analyzed and concluded.

6 RESULTS

The collected results are sorted according to performance metrics for every compaction strategy.

6.1 System Metrics

6.1.1 Disk Utilization

Disk utilization of a system is defined as the amount of load applied on the disk at that instance of time is called disk utilization.

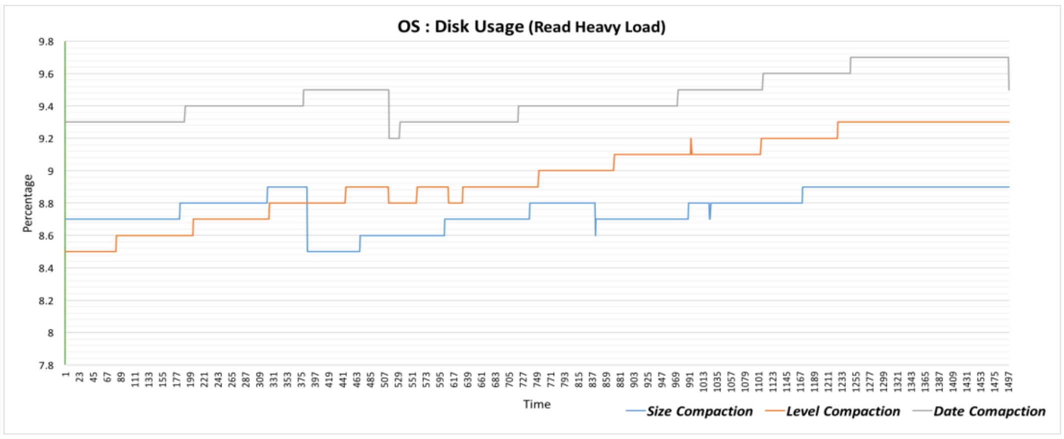

Figure 2 Disk usage read heavy load

In this figure x-axis indicates time measured in seconds and y-axis indicates percentage.

The disk utilization of size compaction and leveled compaction has similar behavior. But date tiered compaction has a very different state of behavior. From the log files created while testing, we observed that date tiered compaction has higher disk utilization among three in the beginning and kept consistently increasing.. While disk utilization in sized compaction and leveled compaction kept consistently increasing in read heavy load.

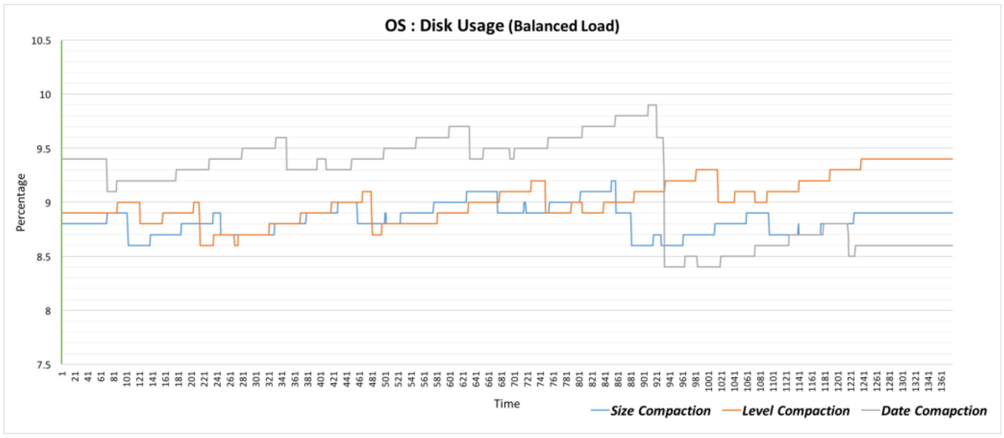

Figure 3 OS disk usage balanced work load

In this figure x-axis indicates time measured in seconds and y-axis indicates percentage.

The disk utilization of size tiered compaction and leveled compaction has similar behavior. But date tiered compaction has a very different state of behavior. From the log files created while testing, we observed that date tiered compaction has higher disk utilization among three in the beginning and kept decreasing at the end. While disk utilization in sized compaction and leveled compaction kept consistently increasing in balanced load.

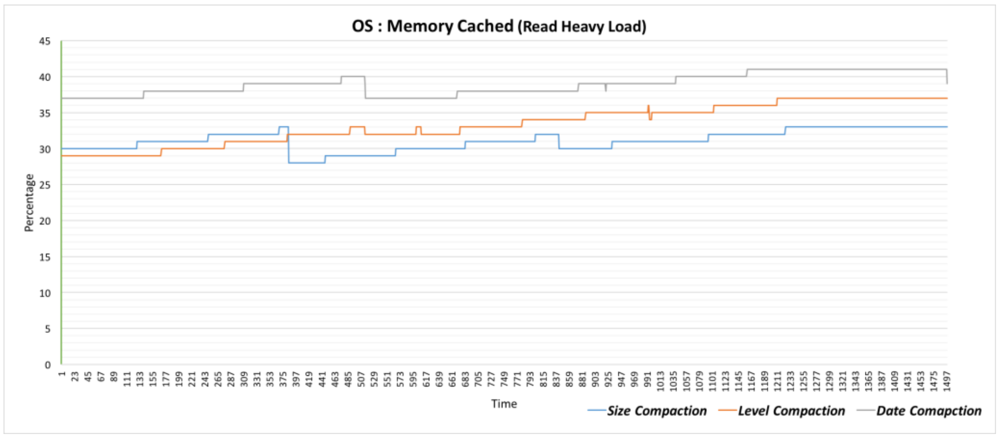

Figure 4 OS memory cached read heavy load

In this figure x-axis indicates time measured in seconds and y-axis indicates percentage.

When a multiple writes are sent to a node, some of the request are cached. Memory is cached different for every compaction strategy.

Memory cached is linearly related to disk utilization. Date compaction started with higher disk utilization than other two and ended with highest of other two. Sized compaction and leveled compaction cached memory was similar but at the end of the test leveled compaction disk utilization was higher among the two.

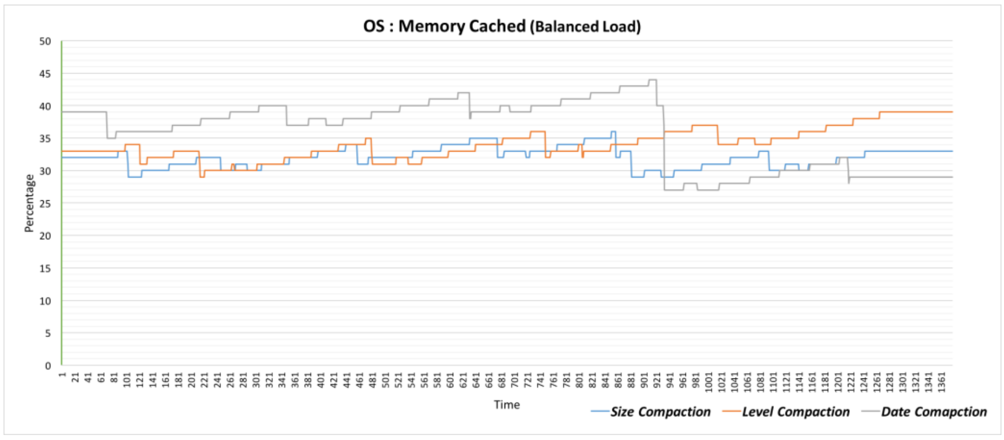

Figure 5 Memory cached for read heavy and balanced work load

In this figure x-axis indicates time measured in seconds and y-axis indicates percentage.

Date compaction started with higher disk utilization than other two and ended with lowest of other two. Sized compaction and leveled compaction cached memory was similar but at the end of the test leveled compaction disk utilization was higher among the two.

6.1.2 Disk Read Rate

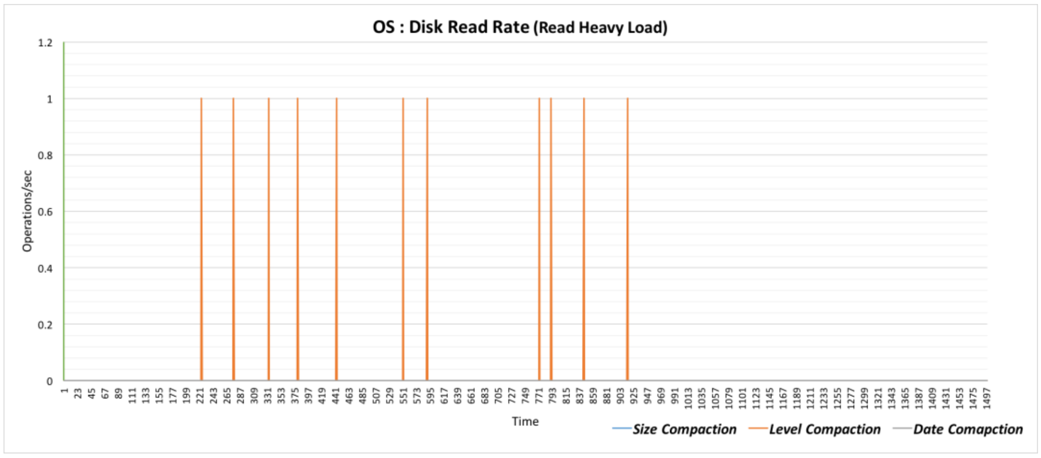

Figure 6 Disk read rate for read heavy load
In this figure x-axis indicates time measured in seconds and y-axis indicates operations/sec.

As the work load is a read heavy not much of write rates are observed. Some of the reads are observed when used leveled compaction. The read rate observed with sized compaction is not high as well.

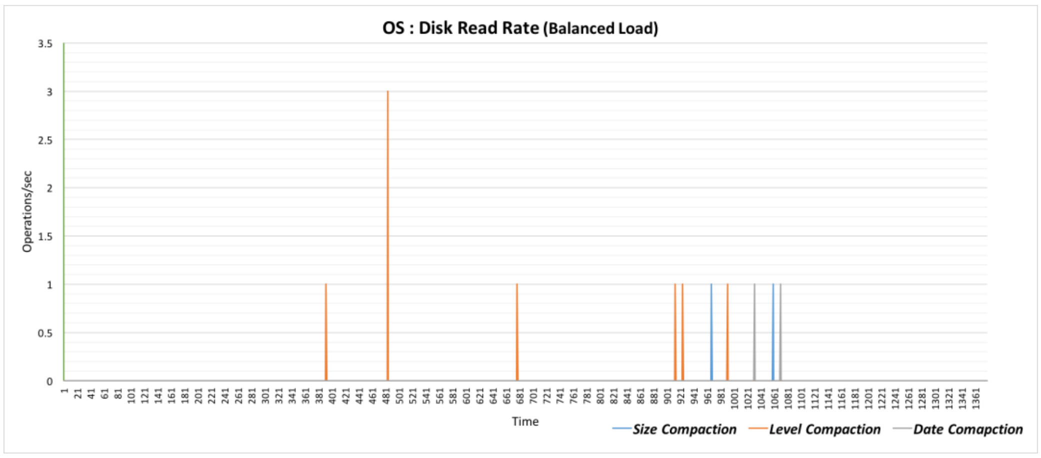

Figure 7 Disk read rate balanced load
In this figure x-axis indicates time measured in seconds and y-axis indicates operations/sec.

As the work load is balanced. Some of the reads are observed when used size compaction. The read rate observed with sized compaction is not high as well. Read rate observed in leveled is higher compared to size compaction.

6.1.3 Disk write throughput

1. In Disk Write Throughput Graph we can see that LTCS is uniform in Disk write operation, as it writes to memtables at a given time maintain small

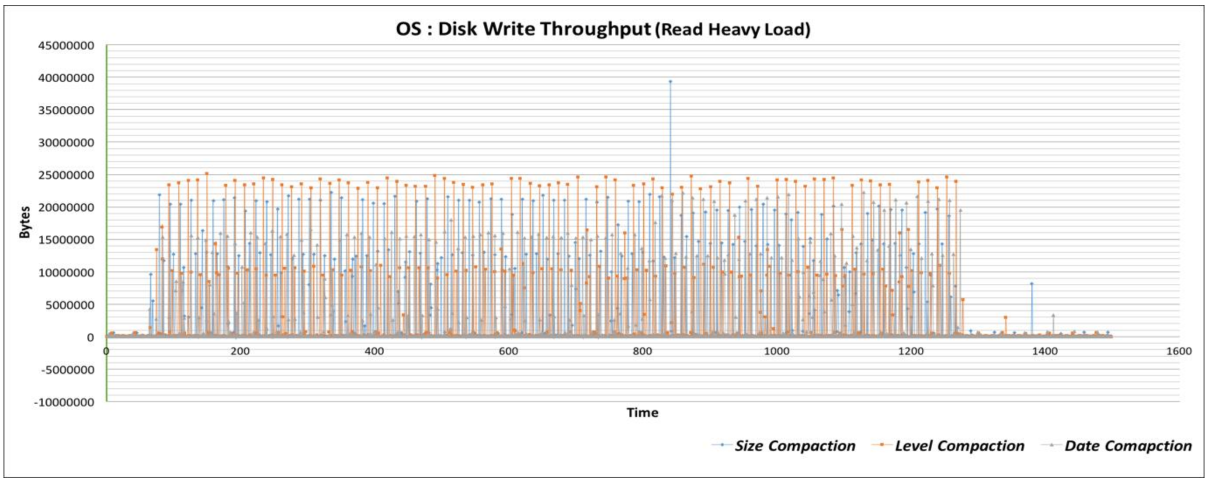

Figure Disk write throughput read heavy load
In this figure x-axis indicates time measured in seconds and y-axis indicates bytes.

quantity of data before flushing it out to SSTables for Compacting, which in turn results in LCS have more Disk Read, which can be seen in Disk Read T

6.2 Cassandra Metrics

6.2.1 Operations/second

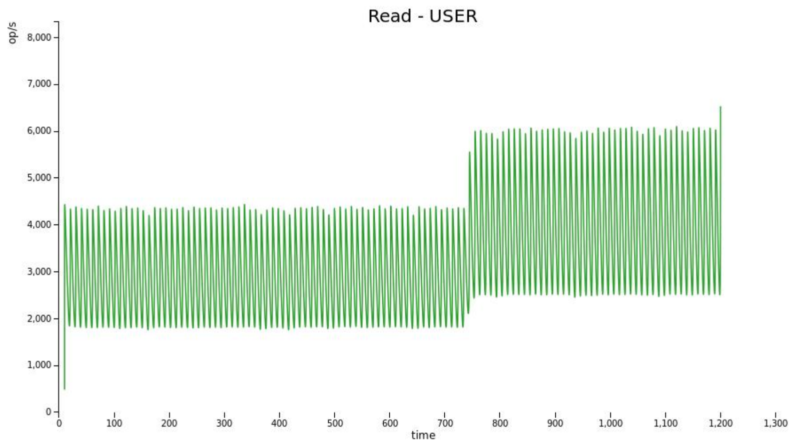

Figure 8 Operation rate for date tired compaction strategy
In this figure x-axis indicates time measured in seconds and y-axis indicates operations/sec.

The operations per second in date tiered are very low in the beginning and high by the end of the test this might be because of the warm up time Cassandra took before reaching highest operation rate.

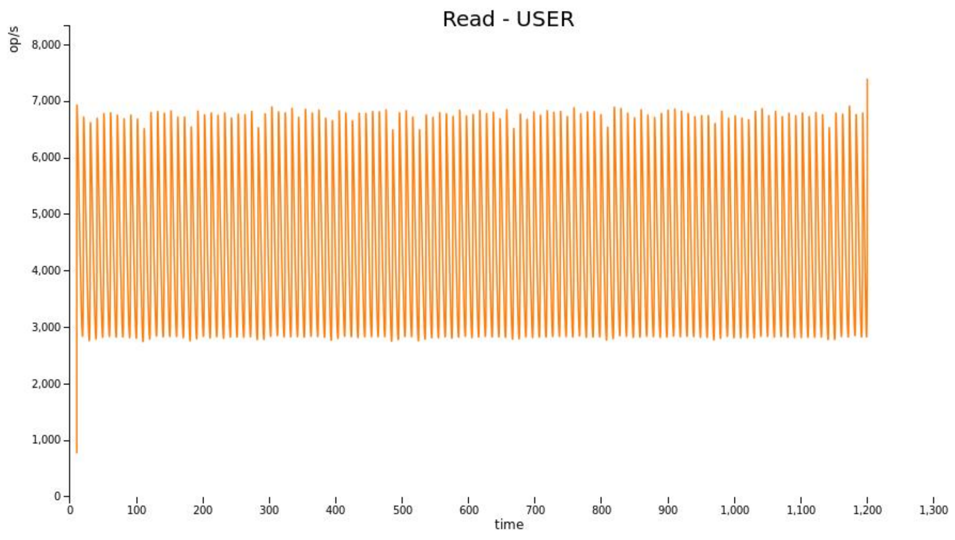

Figure 9 Operation rate for leveled compaction strategy
In this figure x-axis indicates time measured in seconds and y-axis indicates operations/sec.

The operations/second in leveled throughout the test was very low in comparison with date tiered compaction. But it was consistently performing throughout the test and does not have war up delay.

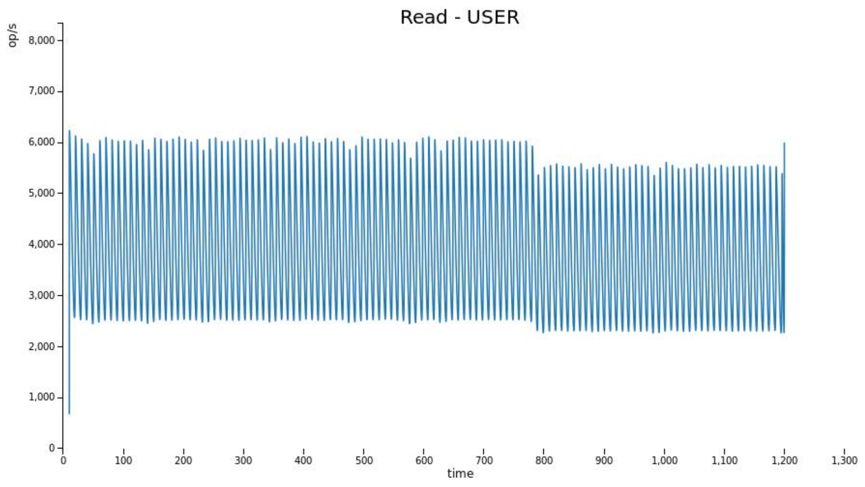

Figure 10 Operation rate for size tiered compaction strategy
In this figure x-axis indicates time measured in seconds and y-axis indicates operations/sec.

size tiered compaction the operations handled in a second are very high when compared to other two. There is a slight decrease at the end of the test but still had higher than date tiered compaction.

6.2.2 Number of Compactions

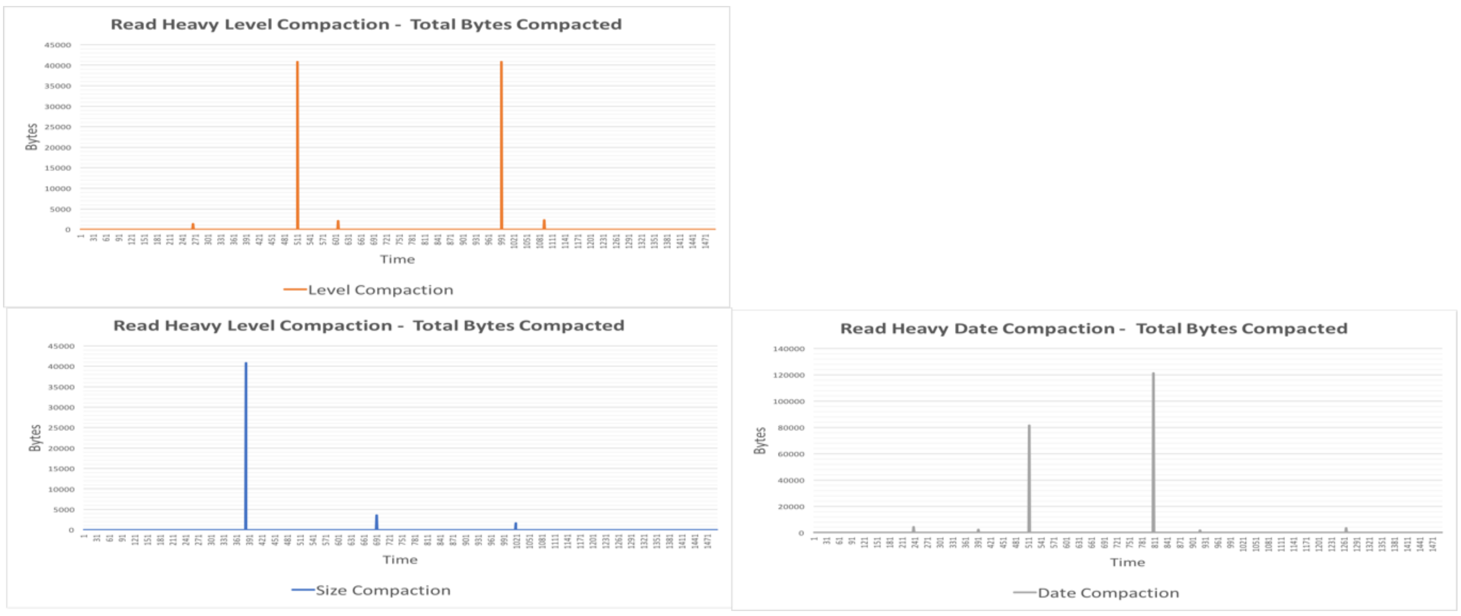

Figure Number of compactions read heavy work load
In this figure x-axis indicates time measured in seconds and y-axis indicates bytes.

Even though leveled compactions have less operation times the number of bytes compacted in leveled compactions are significantly higher. Sized compaction showed very low bytes compacted even though it had ability to handle higher operations rate. Considering these two meters date tiered compaction is a better compaction strategy because even tough its ability to handle operation rate is lower than sized compaction. It out performs size compaction in total number of bytes compacted.

6.2.3 Mean Latency

1. Latency mean graph reflects the above point for DTCS. Also, we can see that LTCS performs best in Read Heavy work load.

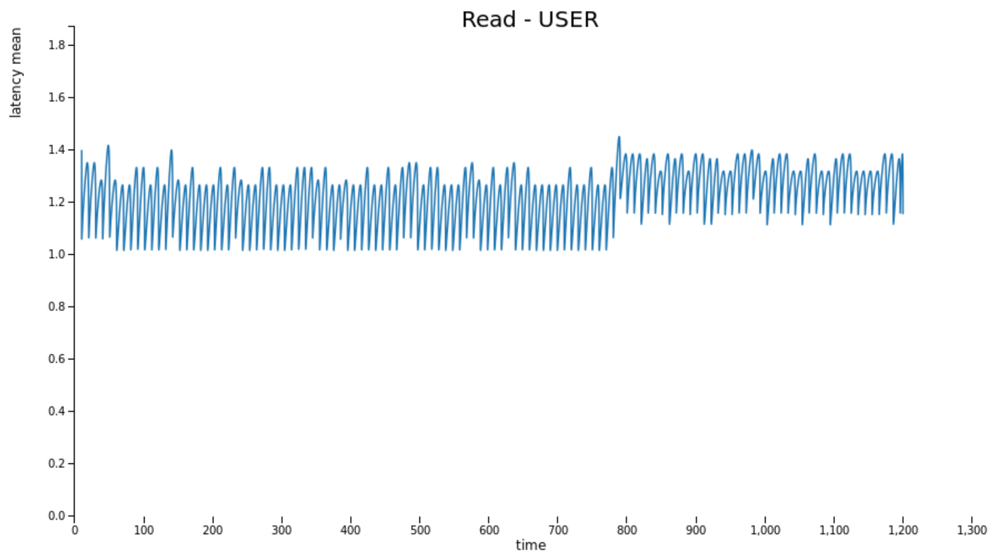

Figure 11 Size tired latency mean graph
In this figure x-axis indicates time measured in seconds and y-axis indicates latency mean in seconds.

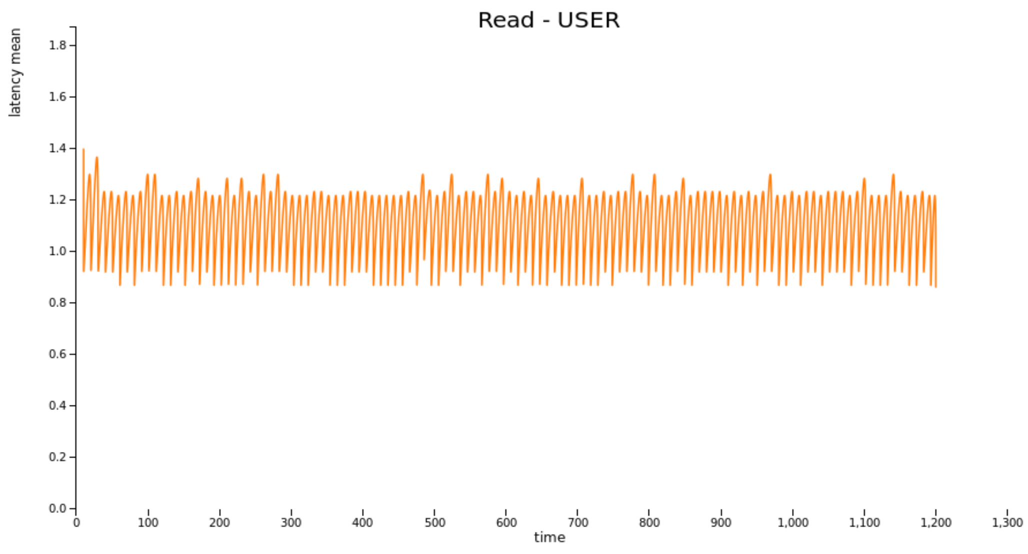

Figure 12 Leveled tiered latency mean graph
In this figure x-axis indicates time measured in seconds and y-axis indicates latency mean in seconds.

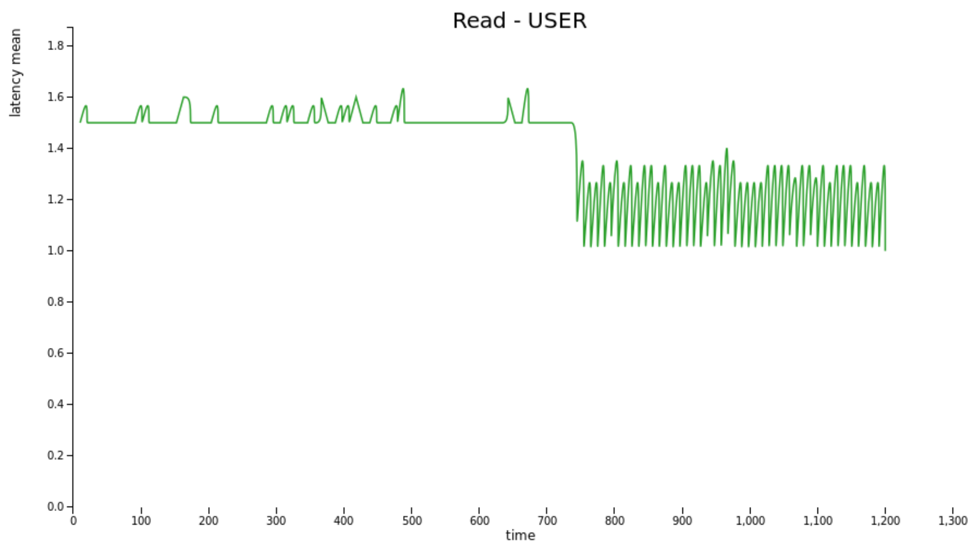

Figure 13 Date tiered latency mean graph
In this figure x-axis indicates time measured in seconds and y-axis indicates latency mean in seconds.

6.3 Balanced

6.3.1 Operations/sec

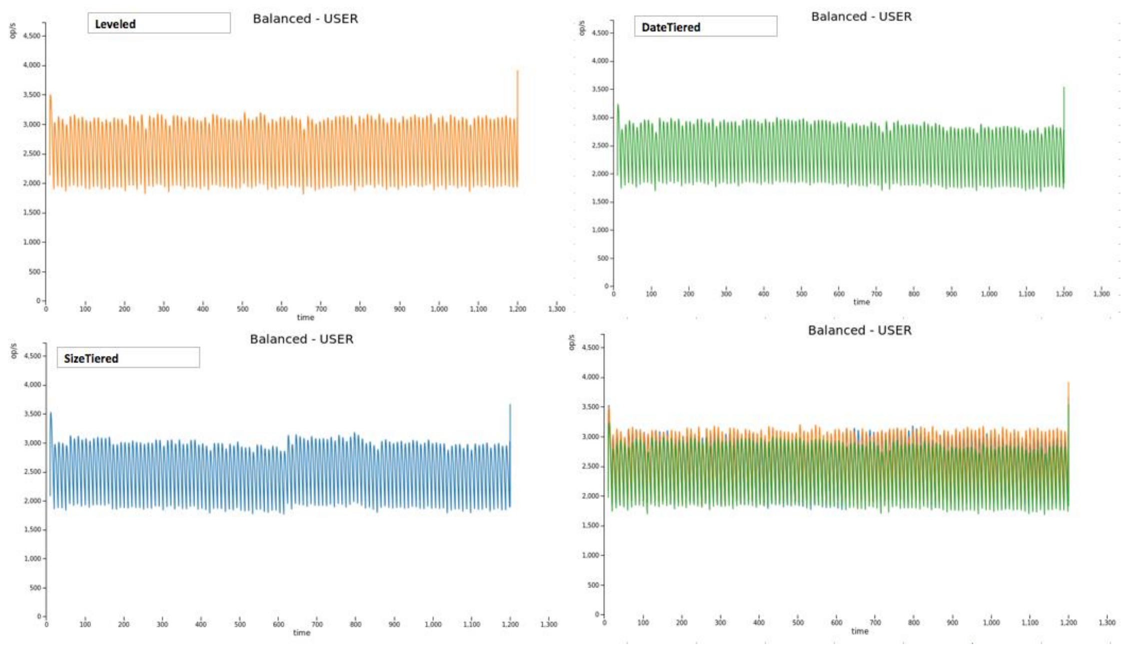

Figure 14 Operations per seconds graphs for balance work load
In this figure x-axis indicates time measured in seconds and y-axis indicates bytes.

Even though the operations/sec in three of them are similar leveled handles a little more operation rate than rest of the two. The operations rate improved white compared to read heavy this is because they could even handle write request as good as read heavy.

6.3.2 Total Bytes Compacted

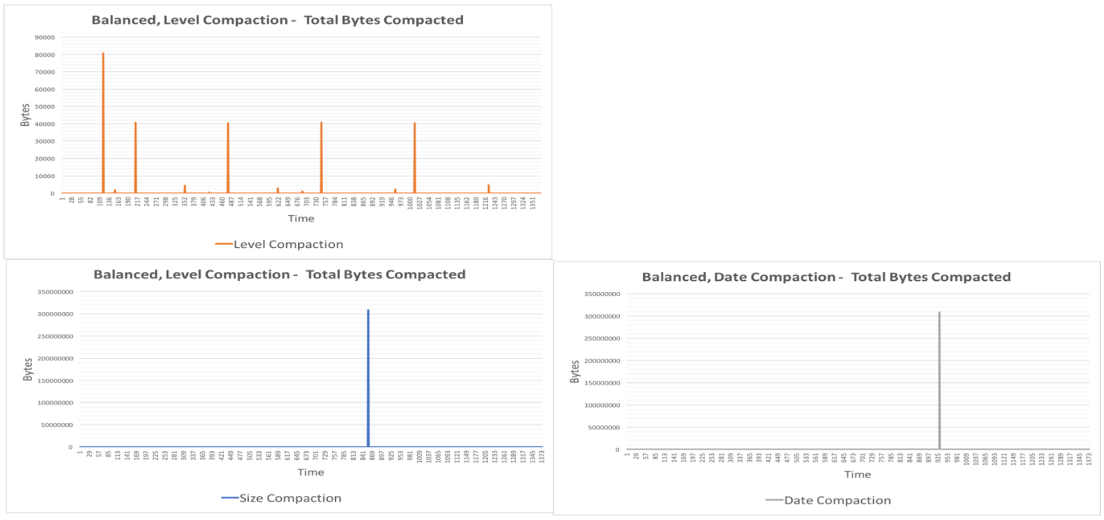

Figure 15 Total bytes compacted
In this figure x-axis indicates time measured in seconds and y-axis indicates bytes.

Even though number of bytes compacted in leveled compaction the total amount of bytes compacted in a single compaction is higher in sized compaction and date compaction.

Considering operations /sec and total amount of bytes compacted in Cassandra date tiered compaction has better ten density to handle both operations/sec and total bytes compacted**.**

6.3.3 Latency Mean

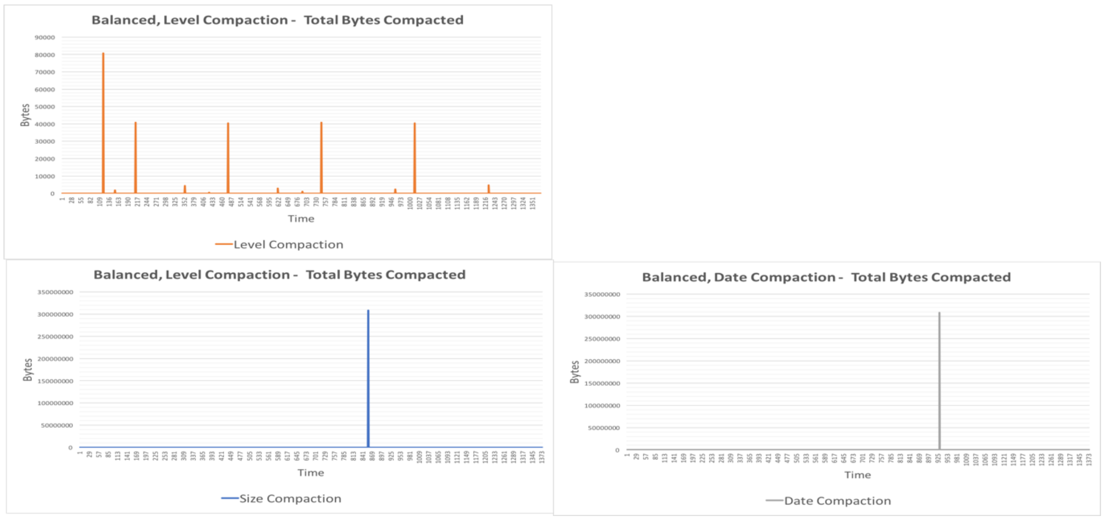

Figure 16 latency mean graph
In this figure x-axis indicates time measured in seconds and y-axis indicates bytes.

Even though number of bytes compacted in leveled compaction the total amount of bytes compacted in a single compaction is higher in sized compaction and date compaction.

Considering operations /sec and total amount of bytes compacted in Cassandra date tiered compaction has better ten density to handle both operations/sec and total bytes compacted.

7 CONCLUSION AND FUTURE WORK

The main aim of the 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.

For read heavy work load, OS load and CPU utilization is equal for all three compaction strategies. For memory cached, date tiered compaction is higher than other two. For read latency and Disk throughput leveled compaction is higher than other two. For total bytes compacted and number of compactions date tiered compaction is better than other two compaction strategies. For disk usage, date compaction is good. Considering the above metrics the performance of Cassandra with date tiered compaction strategy is outstanding. So we conclude that with read heavy workload it is better to use date tiered compaction. There are large number of compactions and heavy data compacted by using date tiered compaction strategy in 20-minute time frame, compared to size tiered and leveled 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.

For balanced work load, OS load and CPU utilization are equal. For memory cached date tiered compaction better than the other two. For Read latency leveled compaction strategy works well. Disk throughput is equal for all three compaction strategies. For total bytes compacted date tiered works well. For number of compactions and disk usage date tired is good when compared to size tiered compaction strategy and leveled compaction strategy. Considering the above metrics the performance of Cassandra with date tiered compaction strategy is outstanding. So we conclude that with balanced workload it is better to use date tiered compaction strategy.

References

[1] "The Apache Cassandra Project." [Online]. Available: http://cassandra.apache.org/. [Accessed: 04-May-2016].

[2] 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.

[3] C. Y. Kan, "Cassandra Query Language," in *Cassandra Data Modeling and Analysis*, Packt Publishing, 2014.

[4] M. Brown, "Preface," in *Learning Apache Cassandra*, Packt Publishing, 2015.

[5] 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.

[6] 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.

[7] "Big Data Platforms: An Overview," 19:00:16 UTC.

[8] C. A. Frangos, "Using Strategy Maps to Prioritize Learning and Performance Improvement Agendas," *Perform. Improv.*, vol. 46, no. 2, pp. 26–29, Feb. 2007.

[9] 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.

[10] C. Y. Kan, *Cassandra Data Modeling and Analysis*. Packt Publishing, 2014.

[11] "WritePathForUsers - Cassandra Wiki." [Online]. Available: https://wiki.apache.org/cassandra/WritePathForUsers. [Accessed: 11-May-2016].

[12] A. Rafique, *Evaluating NOSQL Technologies for Historical Financial Data*. 2013.

[13] S. Abbireddy, *A Model for Capacity Planning in Cassandra : Case Study on Ericsson's Voucher System*. 2015.

[14] "Yahoo Cloud Serving Benchmark | labs.yahoo.com." [Online]. Available: https://labs.yahoo.com/news/yahoo-cloud-serving-benchmark. [Accessed: 11-Feb-2016].

[15] V. Johansen, *Object serialization vs relational data modelling in Apache Cassandra: a performance evaluation*. 2015.

[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] N. S. Kuruganti, *Distributed databases for Multi Mediation : Scalability, Availability &amp;amp; Performance*. 2015.

[18] S. Thorsen, *Replica selection in Apache Cassandra : Reducing the tail latency for reads using the C3 algorithm*. 2015.

[19] C. Hamberg and D. Persson, *Storskalig analys och lagring av rådata i datormolnet*. 2012.

[20] N. Neeraj, *Mastering Apache Cassandra*. Packt Publishing Ltd, 2013.

[21] "OpsCenter and DevCenter | DataStax Academy: Free Cassandra Tutorials and Training." [Online]. Available: https://academy.datastax.com/downloads/ops-center. [Accessed: 18-May-2016].