Impact of Cassandra Compaction on Dockerized Cassandra’s performance
Using Size Tiered Compaction Strategy

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

Contact Information:
Author(s):
Biswajeet Mohanty
E-mail: bimo15@student.bth.se

University advisor:
Emiliano Casalicchio
Department of computer science and engineering

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

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

Context. Cassandra is a NoSQL Database which handles large amount of data simultaneously and provides high availability for the data present. Compaction in Cassandra is a process of removing stale data and making data more available to the user. This thesis focusses on analyzing the impact of Cassandra compaction on Cassandra’s performance when running inside a Docker container.

Objectives. In this thesis, we investigate the impact of Cassandra compaction on the database performance when it is used within a Docker based container platform. We further fine tune Cassandra’s compaction settings to arrive at a sub-optimal scenario which maximizes its performance while operating within a Docker.

Methods. Literature review is performed to enlist different compaction related metrics and compaction related parameters which have an effect on Cassandra’s performance. Further, Experiments are conducted using different sets of mixed workload to estimate the impact of compaction over database performance when used within a Docker. Once these experiments are conducted, we modify compaction settings while operating under a write heavy workload and access database performance in each of these scenarios to identify a sub-optimal value of parameter for maximum database performance. Finally, we use these sub-optimal parameters to perform an experiment and access the database performance.

Results. The Cassandra and Operating System related parameters and metrics which affect the Cassandra compaction are listed and their effect on Cassandra’s performance has been tested using some experiments. Based on these experiments, few sub-optimum values are proposed for the listed metrics.

Conclusions. It can be concluded that, for better performance of Dockerized Cassandra, the proposed values for each of the parameters in the results (i.e. 5120 for Memtable_heap_size_in_mb, 24 for concurrent_compactors, 16 for compaction_throughput_mb_per_sec, 6 for Memtable_flush_writers and 0.14 for Memtable_cleaup_threshold) can be chosen separately but not the union of those proposed values (confirmed from the experiment performed). Also the metrics and parameters affecting Cassandra performance are listed in this thesis.

Keywords: Docker, Cassandra, Cassandra compaction, NoSQL database
LIST OF FIGURES

Figure 1: Data model of Cassandra ................................................................. 8
Figure 2: Cassandra Storage Engine [13] ....................................................... 9
Figure 3: (a) Hypervisors based virtualization, (b) Docker Container based virtualization[19] ................................................................. 11
Figure 4: Three stages of Literature review process [30] .................................. 18
Figure 5: Testing Environment ....................................................................... 19
Figure 6: Design of the test environment ........................................................ 20
Figure 7: Total number of compactions for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ............................................ 32
Figure 8: Total number of bytes compacted for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ........................................ 33
Figure 9: Total number of completed tasks for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ........................................ 33
Figure 10: Total number of pending tasks for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ........................................... 34
Figure 11: Disk usage percentage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ......................................................... 35
Figure 12: CPU usage percentage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ......................................................... 35
Figure 13: Memory usage percentage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ......................................................... 36
Figure 14: Latency mean for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ................................................................. 36
Figure 15: Bytes compacted for Memtable_heap_space_in_mb ............................ 38
Figure 16: Compaction count for Memtable_heap_space_in_mb .......................... 38
Figure 17: Bytes compacted for Memtable_flush_writers .................................... 39
Figure 18: Compaction count for Memtable_flush_writers .................................. 40
Figure 19: Bytes compacted for concurrent_compactors ..................................... 41
Figure 20: Compaction count for concurrent_compactors .................................. 41
Figure 21: Bytes compacted for Compaction_throughput_mb_per_sec .................. 42
Figure 22: Compaction count for Compaction_throughput_mb_per_sec ................ 42
Figure 23: Comparison of bytes compacted for sub-optimal parameters (when used together) and default parameters ................................................. 44
Figure 24: Comparison of compaction count for sub-optimal parameters (when used together) and default parameters ................................................. 44
LIST OF TABLES

Table 1: Dependent and independent variables for answering RQ3 and RQ4 ............................. 20
Table 2: compaction parameters that impact Cassandra's performance ..................................... 27
Table 3: Number of threads ........................................................................................................ 31
Table 4: Chosen Compaction and OS metrics ............................................................................. 31
Table 5: Average and RMSE values of compaction metrics for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ......................................................... 34
Table 6: Average and RMSE values of OS metrics for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads ................................................................. 37
Table 7: Average and RMSE values of Memtable_heap_space_in_mb for bytes compacted and compaction count .................................................................................................. 39
Table 8: Average and RMSE values of Memtable_flush_writers for bytes compacted and compaction count ......................................................................................................... 40
Table 9: Average and RMSE values of Concurrent_compactors for bytes compacted and compaction count ................................................................................................................. 41
Table 10: Average and RMSE values of Compaction_throughput_mb_per_sec for bytes compacted and compaction count .................................................................................................................. 43
Table 11: Sub-optimal value of each compaction parameter .......................................................... 43
Table 12: Average and RMSE values of sub-optimal parameters (when used together) and default parameters for bytes compacted and compaction count ............................................. 44
Table 13: Sub-optimal value of each compaction parameter .......................................................... 48
TERMINOLOGY AND ACRONYMS

1. **CPU**: Central Processing Unit
2. **OS**: Operating System
3. **RAM**: Random Access Memory
4. **HPC**: High Performance Computing
5. **SQL DB**: Structured Query Language Database
6. **NoSQL DB**: Not only Structured Query Language Database
7. **VM**: Virtual Machine
8. **VMM**: Virtual Machine Monitor
9. **SSTables**: Sorted String Tables
10. **STCS**: Size Tiered Compaction Strategy
11. **LCS**: Leveled Compaction Strategy
12. **DTCS**: Date Tiered Compaction Strategy
13. **KPI**: Key Performance Indicator
14. **RMSE**: Root Mean Square Error
15. **RQ**: Research Question
1 INTRODUCTION

New advanced technologies like distributed databases (all NoSQL DB) are capable of handling big data with their inherent schema free data model and horizontal scalability features to store and retrieve the data quickly [1]. The popular NoSQL databases available are Facebook’s Cassandra [2], Mongo DB [3], Oracle’s NoSQL DB [4], Amazon’s Dynamo [5], Google’s Big Table [6] and Apache’s Hbase [7].

NoSQL stands for Not Only Structured Query Language. NoSQL databases are distributed and non-relational databases, where they are not primarily built on tables. They can store huge amounts of data and can process the stored data in parallel across a large number of commodity servers. They use non-SQL languages to query and manipulate the data. [8]

NoSQL databases are divided into five different types based on their characteristics and data storage models: wide column store/column family, document store, key value/tuple store, consistent key value store and graph databases. Cassandra is a type of ‘column store/column family’ NoSQL database with a distributed storage system developed by Facebook for storing and managing huge amounts of unstructured data [9]. It has become increasingly popular with many big organizations today like Yahoo, Facebook etc. which are moving towards migrating to Cassandra DB for their data consolidation needs and many other leading IT firms which are developing data mining applications for big data on Cassandra platform [10]. Cassandra does not have a fixed schema like traditional relational databases and it can have any number of columns in a row contrary to single row limitations with relational databases [11]. It's popularity today has been elevated due to its reliability, scalability and robustness features [12]. It uses a process called compaction to store the data into the disks called SSTables, which are immutable in nature [13]. Compaction plays an important role in the performance of Cassandra as huge amount of data gets stored into the SSTables by using one of the compaction strategies (i.e. Size Tiered Compaction Strategy, Leveled Compaction Strategy and Date Tiered Compaction Strategy) and response time of any read/write request to the Cassandra is affected by the compaction process.

In this thesis we test the impact of Cassandra compaction when running inside a Docker. According to the definition of a Docker mentioned on their website, “Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment” [15]. Essentially, a Docker allows a software to execute itself without the need of an operating system environment, providing it with the necessary utilities to do its intended task. This setup increases the software performance as all the available resources are used by the software to ensure its successful execution.

1.1 What is Cassandra?

Cassandra is a NoSQL database and a distributed storage system developed by Facebook to store and manage huge amounts of unstructured data [9]. Cassandra differs from the relational database in the table rows schema. There is no fixed schema for the Cassandra database, that is, there can be any number of columns in a row [11]. It is designed to run on top of many nodes. But there are chances of components of a
node to fail. Cassandra solves this problem by running in a steady state, showing its robustness. Cassandra also has the feature of reliability (design system with cheap hardware) and scalability (user handles the design and configuration of data) [12].

Cassandra consists of a yaml file called cassandra.yaml, which contains all the settings of the parameters, even the compaction parameters that are used by Cassandra. This yaml file can be edited to tune the parameters. After making changes to the .yaml file, the Cassandra process needs to be restarted to bring the changes into effect.

1.1.1 Cassandra Data Model

The Cassandra data model consists of different parts. The different parts are as follows: column, key, rows, column family, keyspace and cluster. Column is the smallest part of the data stored. Each column has three different attributes namely column name, column value and timestamp. Each column has a specific key and each key is represented as a row. There are many rows and each row has a different structure. All these rows combine to form a column family and these rows inside a column family are placed by using a partitioning strategy. These column family form a keyspace. These keyspaces act as a database and all the keyspaces combine together to form a cluster. [9]

![Figure 1: Data Model of Cassandra](image)

1.1.2 Cassandra Architecture

The architecture of Cassandra uses a peer-to-peer networking to communicate with the other Cassandra machines or the nodes, which are connected in a cluster. There is continuous communication between the nodes at all the times. It has the features of replication and automatic synchronization of data across all the nodes, that is, any changes to the data in a node is replicated to the other nodes in the cluster and is automatically synchronized. Due to the replication, there is no single point of failure and there is no data loss if any of the nodes fail. Any node in the cluster can be set as a seed node and the client directly communicates with the seed node to perform the operations. The seed node is called as the coordinator to transmit the client’s request to the other nodes in the cluster. [9]
1.1.2.1 Cassandra Storage Engine

Cassandra stores the data on the disk by using a mechanism similar to Log-Structure Merge (LSM) trees [13]. The different components used to store the Cassandra data are as follows:

1. **CommitLog**: helps Cassandra in achieving its durable nature, that is, any change in data or a write operation is first stored/written to the commitlog, which stores it permanently. There is no chance of data loss if any system failure occurs. [13]
2. **MemTable**: It is similar to the cached data, which is sorted by keys. They don’t have any replica and any changes to the existing data is overwritten if the same key is used. [13]
3. **SSTable**: is the disk representation of the data. It is immutable in nature. [13]

As shown in Figure 2, any write operation first writes/append the data to the CommitLog and then updates the MemTable. MemTable stores the the data based upon some criteria and then flushes the data to the immutable SSTables after the criteria is fulfilled. Now the CommitLog gets rid of the data, which was stored earlier and again starts storing new data. [13]

![Figure 2: Cassandra Storage Engine](image)

1.1.3 Cassandra Compaction

Compaction is the process of merging the immutable SSTables to form a new and single SSTable. As the data gets stored on to the different SSTables, any read operation to the data requires reading data from the different SSTables. This problem is overcome by doing the compaction. There are different types of compaction strategies like size-tiered compaction strategy (STCS), level compaction strategy (LCS) and Date-tiered compaction strategy (DTCS). But, STCS is the default strategy used in the Cassandra compaction. [13][14]

1.1.3.1 Size Tiered Compaction Strategy (STCS)

It is the default compaction strategy in the Cassandra yaml file. In this the SSTables of the similar size are merged together to form a new and larger SSTable.
Then these larger SSTables of similar size are merged to form even larger SSTables. It works good for workloads with write heavy operations. But, it holds onto the stale data for long time, due to which the amount of memory required increases. [13][14]

1.1.3.2 Leveled Compaction Strategy (LCS)
This compaction strategy can be used for compaction by changing the settings in the cassandra.yaml file. The compaction process works in levels. The data from the MemTable gets stored into the SSTables in the level one (L0). These SSTables are merged with the larger SSTables at level two (L1) by compaction. This process of merging the SSTables carries on to next level and so on. It works good for workloads with read heavy operations. But, the operation latency gets affected by the higher I/O (input/output) utilization. [13][14]

1.1.3.3 Date Tiered Compaction Strategy (DTCS)
This compaction strategy can be used for compaction by changing the settings in the cassandra.yaml file. This strategy is similar to the STCS, but the compaction process merges the SSTables written in the same time frame. It uses Time-To-Live (TTL) timestamps to merge the SSTables of similar ages. It works good for compacting the time series data. [14]

1.1.4 Cassandra Stress Tool

According to the definition given in the Datastax documentation, “Cassandra-stress tool is a java-based stress testing utility for basic benchmarking and load testing a Cassandra cluster”. It is used to populate the Cassandra cluster with huge amount of data and stress testing the Cassandra tables and queries. A single experiment should be conducted several times to find the issues in the data model. The data gets stored into the keyspaces and tables, when a stress test is done. The default keyspace to store the data is ‘keyspace1’ and, the default tables to store the data is ‘standard1’ or ‘counter1’ depending upon the type of the table being tested. [16]

1.2 What is Docker?

Docker is a container-based virtualization technique. Each container can store an application software or the business service software with all the system libraries and tools required to run the software. Containers share the same kernel of the operating system, when running on the same machine. They start instantly and the usage of RAM (Random Access Memory) is very less. Different applications can run on different containers simultaneously and, the processes are isolated and protected from each other. [15]

Lately Docker is being used as a popular tool for managing Linux containers. Research in the field of containers reveal that when virtual machines are compared with bare metal, the processing speed and throughput are almost equal. This makes Docker a silver lining in the cloud full of virtual machines. Docker is used in testing purposes for testing applications in an isolated state. Docker containers are implemented in Linux using ‘cgroups’, which allows resources such as CPU and memory to be shared among them. The containers are isolated and have a different kernel namespace. [17]
1.2.1 Hypervisors vs Docker Containers

Hypervisors technique is the most commonly used virtualization technique. In this technique, there is a virtual machine monitor (VMM) on top of the host operating system (OS) [18]. VMM controls different virtual machines (VM), where each VM consists of its own OS. This is the traditionally used virtualization technique. But, it causes a high performance overhead, while it is used for HPC [18]. In Hypervisors, a guest operating system is required to be installed, before downloading the application and, its supporting binaries and libraries [15]. Whereas in Containers, we only need to download the application and all the dependent binaries and libraries. There is no need of installing an Operating System (OS) [15]. Figure 3 gives an idea of the working of the hypervisors and the containers.

![Figure 3: (a) Hypervisors based virtualization, (b) Docker Container based virtualization][19]

1.3 Scope

The scope of the thesis is limited to analyzing the impact of Cassandra compaction on Cassandra’s performance when used inside a Docker container using Size Tiered Compaction Strategy (STCS) on a single node or a single machine. Experiments are conducted using different workloads and the output is a graphical representation of OS metrics (CPU usage, Disk usage etc.) vs time and Cassandra metrics (Bytes compacted, completed tasks etc.) vs workloads to better analyze the impact of compaction. Also the thesis is limited to tuning the compaction parameters one at a time and analyzing the results to find a sub-optimal value of each compaction parameter only for a write heavy workload (i.e. 90%writes and 10% reads), when used on a Docker based platform. Output yielded from the experiments are represented in the form of graphs to better analyze the results and find an sub-optimal compaction setting for a write heavy workload, when used on a Docker based platform.
1.4 Problem Description and Motivation

Cassandra and Docker are the leading contenders in the field of NoSQL databases and Container technology respectively. Cassandra stores its data into the disks by the process of compaction. And, while measuring the performance of Cassandra, compaction plays a very important role. Compaction statistics are the Key performance indicators/markers (KPI/KPM) for Cassandra. Docker, a container technology is in forefront of today's virtualization over big data cloud computing. But, a very little research has been done on the integration of Cassandra with Docker. Important questions such as, "what is the impact when Cassandra runs inside a Docker container?" and "How is the Cassandra’s performance affected in a Docker environment?" is what we are trying to assess and have answers for! Finding answers to these questions is what has motivated us to embark on this research thesis wherein we need to integrate Cassandra on top of Docker and then investigate the impact of Cassandra compaction on Cassandra’s performance.

1.5 Aim and Objectives

1.5.1 Aim

The aim of the project is to:
1. Run Cassandra inside Docker container
2. Investigate the impact of Cassandra compaction (with the default and tuned parameters) on Cassandra’s performance working within a Docker
3. Identify KPI (Key Performance Indicator) to access the performance

1.5.2 Objectives

The objectives of this thesis is to:
1. Learn about the Cassandra database and the Cassandra compaction
2. Learn about Docker containers
3. Study the impact of Cassandra compaction when Cassandra runs inside a Docker container
4. Do a literature review for finding the different performance metrics and parameters on which the Cassandra compaction is dependent
5. Learn and study about the different performance parameters related to Cassandra compaction
6. Learn and study about the different performance metrics (output) related to Cassandra compaction
7. Perform experiments with default Cassandra settings using different workloads and analyze the results to find out the impact of Cassandra compaction
8. Perform experiments by tuning the compaction parameters and by using a write heavy workload, and then analyze the results to find the sub-optimal value of Cassandra compaction parameters to be used inside a Docker container

1.6 Research Questions

This section describes about the research questions that are needed to be answered to execute a successful research.

RQ1: What are the performance metrics that better adhere to measure the effect of compaction on the Cassandra performance for a Docker based platform?
This question is framed to learn about the different Cassandra compaction metrics, which can be used to measure the performance of Cassandra running inside a Docker container. This is solved by doing a Literature review.

**RQ2:** What are the Cassandra compaction parameters that allow to tune the Cassandra performance for a Docker based platform?

This question is framed to learn about the different parameters, that are directly or indirectly related to Cassandra compaction. These parameters may affect the performance of Cassandra, when tuned. This is solved by doing a Literature review.

**RQ3:** What is the impact of Cassandra compaction (using the default settings of Cassandra compaction) on the Cassandra performance for a Docker based platform, when different workloads (1) 90%writes and 10%reads, (2) 50%writes and 50%reads, and (3) 70%writes and 30%reads are considered?

This question is framed to investigate the impact of compaction on Cassandra’s performance on a Docker platform by using the default settings of Cassandra. This is solved by performing experiments using different workloads.

**RQ4:** What is the impact of Cassandra compaction (by tuning each compaction parameter) on the Cassandra performance for a Docker based platform, when 90%writes and 10%reads workload is considered?

This question is framed to investigate the impact of compaction on Cassandra’s performance on a Docker platform by tuning each parameter at a time. Then sub-optimal value of the parameters are collected, which can offer better performance of Cassandra on a Docker platform. This is solved by tuning the parameters one at a time and performing experiments on the tuned setting for a write-heavy workload (i.e. 90%writes and 10%reads).

### 1.7 Contribution

The main contribution of the thesis is to understand the effect of compaction when Cassandra runs inside a Docker container. First, the experiments are conducted by using the default setting of Cassandra compaction and using the default compaction strategy, that is, Size Tiered Compaction Strategy (STCS), with different workloads to find the impact of Cassandra compaction on the Cassandra’s performance, for a Docker based platform. The results are shown as a graphical representation of OS metrics (CPU usage, Disk usage etc.) vs time and Cassandra metrics (Bytes compacted, completed tasks etc.) vs workloads to better analyze the impact of compaction.

Then the compaction parameters are tuned and experiments are carried out to find the sub-optimal values of Cassandra compaction parameters for a write heavy workload (i.e. 90%writes and 10% reads), on a Docker based platform. Graphs are drawn from the results and analysis of these results provides the sub-optimal compaction parameter values.

### 1.8 Outline

The rest of the thesis document is organized as follows. Chapter 2 gives an overview about the related works, that has already been done in the field of research.
Chapter 3 gives a detailed view of how the different research methods are carried out. Chapter 4 gives a detailed view of the results and analysis for the different experiments performed. Chapter 5 gives a detailed view of the discussions including the Threats to validity and answering the RQ’s. Chapter 6 gives an overview about the conclusion and the future work.
2 **RELATED WORK**

This chapter highlights the different works done by other researchers that have ties to our research.

2.1 **Cassandra**

The research on Cassandra is advancing in a quick pace [20]. In [21], authors have discussed about the effect of the replication factor and the consistency strategies for the databases HBase and Cassandra by using the YCSB benchmarking tool. In [22], authors have provided an in-depth analysis of compaction-related performance overhead for HBase and Cassandra. In [23], authors have designed a performance monitoring tool for Cassandra, which they have used on a low end machine (cost effective machines). In [24], authors have estimated a performance model for Cassandra, which shows the relationship between the memory space and the system performance. In [25], authors have discussed about the different background mechanisms that help in improving the performance of data-intensive cloud systems using Cassandra, which has an exceptional read-write performance and the Cassandra compaction strategy, a mechanism which runs in the background and merges the SSTable depending on the compaction strategy used. In [26], author has performed a benchmark test for Apache Cassandra, Apache Lucene and MySQL. Apache Cassandra outperforms others in the benchmark test as it has a high read/write throughput and it can handle larger amount of data than the other databases in comparison.

2.2 **Cassandra Compaction**

In [27], author has compared the performances of different compaction strategies, that is, STCS, LCS and DTCS for different workloads (i.e. 50%writes and 50%reads, and 10%writes and 90%reads). Author has also researched about the different Cassandra metrics and Operating System (OS) metrics required to evaluate the success of different compaction strategies. In [14], Datastax discuss about configuring compaction using the different compaction parameters and also discuss about the different compaction metrics to evaluate the performance of Cassandra. The different Cassandra metrics, OS metrics and Cassandra compaction parameters are as follows:

2.2.1 **Cassandra compaction metrics**

1. **Bytes compacted**: It is the total amount of data that has been compacted from the start of experiment.
2. **Completed tasks**: It is the number of compaction tasks that have been performed/completed from the start of the server.
3. **Pending tasks**: It is the number of compaction tasks that are supposed to be done or that are pending to achieve the desired state.
4. **Total compactions count**: It is the total number of compaction that occur within the specified duration of an experiment.
5. **Compaction rate**: It is the number of compactions performed in a second.

2.2.2 **OS metrics**

1. **Memory usage**: It is the percentage of RAM used while performing each experiment
2. **Memory cached:** It is the amount of memory cached for each experiment
3. **Cpu usage:** It is the percentage of CPU resources being used to perform the experiment
4. **Cpu user:** It is the time, which the CPU provides to the user application or user processes
5. **Cpu privilege:** It is the time, which the CPU performs the I/O operation and other background applications.
6. **Disk usage:** It is the percentage of the secondary memory used to store the data being generated while performing the experiments.
7. **Disk throughput:** It is the throughput in MB for the read/write operations being performed.
8. **Disk latency:** It is the time taken for completion of each read/write request to the disk
9. **Read throughput:** It is the throughput in MB at which the read operation is performed.
10. **Write throughput:** It is the throughput in MB at which the write operation is performed.
11. **Read rate:** It is the number of read operation performed per second.
12. **Write rate:** It is the number of write operations performed per second.
13. **Latency mean:** It is the mean of the time taken by a data packet to send and receive it back to its source.

### 2.2.3 Cassandra compaction parameters

1. **Concurrent compactors:** is the number of compaction processes that run simultaneously on the node or the platform, where Cassandra is running. The default value is ‘1’.
2. **Compaction_throughput_mb_per_sec:** is the throughput, which the compaction process achieves to perform the compaction. The default value is ‘16’.
3. **Snapshot before compaction:** is a backup mechanism used by Cassandra to store the data even after deletion of data from the SSTables. It can be set to ‘true’ or ‘false’, with default as ‘false’. It does not have any influence on the performance of Cassandra.
4. **Compaction_large_partition_warning_threshold_mb:** is used to recorded a warning sign, when a larger partition is created by the compaction process. The default value is ‘100’. It does not have any influence on the performance of Cassandra.
5. **MemTable_flush_writers:** is the number of threads that the memTable flush writers use. The default value is ‘8’.
6. **MemTable_heap_space_in_mb:** is the size of the on-heap memory assigned for memtables. The default value is 1/4th of heap size.
7. **Memtable_offheap_space_in_mb:** is the size of the off-heap memory assigned for memtables. The default value is 1/4th of heap size.
8. **Memtable_cleanup_threshold:** is the ratio used to automatically flush the memtables to the SSTables. It is calculated by the formula ‘1/(1+memTable_flush_writers)’. The default value is ‘0.11’.

### 2.3 Docker and Cassandra

Docker is the trending container based virtualization platform used for the High Performance Computing (HPC), which motivates the scientists in researching about the Docker [28]. In [28], authors have performed experiments to show that container technology is maturing day-by-day and the performance overhead is reducing as a
result. In [29], author has proved that containers consume less power as compared to hypervisors, when analyzing the network performance. But, the power consumption is similar when there is heavy CPU and memory workloads.

Cassandra and Docker are the best contenders in their respective fields and very little research has been done on Cassandra integration with Docker. And while evaluating the performance of Cassandra, the effect of compaction needs to be considered as it plays a very important role. This forms the basis of our thesis to examine the impact of Cassandra compaction on Cassandra’s performance for Docker based platform.
3 METHODOLOGY

This chapter describes about the methods used in the research for answering the research questions.

3.1 Literature Review

The Literature review is performed based on the guidelines followed in [30]. Literature review is defined as “The use ideas in the literature to justify the particular approach to the topic, the selection of methods, and demonstration that this research contributes something new” [30]. Literature is studied to gain basic knowledge about Cassandra, Cassandra compaction and Docker. More literature is reviewed for two things: first, studying the different performance metrics to measure the Cassandra performance, and second, studying the different compaction parameters, that can be tuned to enhance the Cassandra performance. Literature review is performed to answer the research question RQ1 and RQ2. The result of RQ1 is provided in the Section 2.2.1 and Section 2.2.2. The result of RQ2 is provided in the Section 2.2.3. The literature review is carried out in a 3-step process: Input, Processing and output as shown in the Figure 4.

Figure 4: Three stages of Literature review process [30]

The three step process for performing the Literature review is as follows:

1. The input to the system should be a relevant literature to the field of research, which is in accordance to the rules or standards for right conduct or practice [30]. This includes, searching the profound databases such as Inspec, Google Scholar, ACM digital library, IEEE Explorer etc. for a relevant literature and also to search for some related documentation in the research field.

2. The Processing step includes:

   a. Know the literature: It includes activities such as listening, defining, describing and identifying [30]. We try to understand the available literature.

   b. Comprehend the literature: It includes activities such as summarizing, differentiating, interpreting and contrasting [30]. We try to understand the significance of the research work done in the available literature.

   c. Apply the literature: It includes activities such as demonstrating, illustrating, solving, relating and classifying [30]. We relate the literature to the concepts in our research.

   d. Analyze the literature: It includes activities such as separating, connecting, comparing, selecting and explaining [30]. We separate
the literature from the other literatures, which are of more importance for our research.

e. Synthesize the literature: It includes activities such as combining, integrating, modifying, rearranging, designing, composing and generalizing [30]. We rearrange the literature gathered from the previous steps, for each of the concepts required in our research.

f. Valuate the literature: It includes activities such as accessing, deciding, recommending, selecting, judging, explaining, discriminating, supporting and concluding [30]. We conclude the literature review by explaining all the theories and concepts, that are required in our research.

3. The output of the literature review process is a well-defined and a straightforward piece of writing. It has a logical structure, and includes precise and sufficient information required to make a reader understand the subject. [30]

Literature review is also conducted on the Datasat documentation on Cassandra [14], to get used to the different concepts about Cassandra and also to find the different compaction related metrics and compaction related parameters. Literature review is also conducted on the documentation of Docker [15], to understand the working of the Docker containers and also to find the different commands required to make the container work.

3.2 Experimentation

Literature review gives us an idea of how to perform the different experiments required for the research. This also helps in designing the experiments and in answering the research question RQ3 and RQ4. The experimentation process is as follows.

3.2.1 Testing Environment

![Figure 5: Testing Environment](image)

Blekinge Institute of Technology administration has provided two different nodes or machines, along with their SSH access. Each of the nodes has 279.4 GB of disk memory, 24GB of RAM, 12 cores and each core has 2 threads (theoretically making it 24 cores). Each node is installed with Ubuntu 14.04.3 LTS. One of the nodes is used as a load generator (Node1) and the other node, as a single node cluster (Node2) to perform the experiments. These two nodes are accessed by SSH tunneling from the
master node as shown in the Figure 5. This thesis is an initial step towards research on Cassandra, Cassandra compaction and Docker. So, single node cluster is used.

3.2.2 Dependent and independent variables

The dependent and independent variables of the experiments are mentioned below:

3.2.2.1 Dependent Variables

The dependent variables of the experiments (for answering RQ3 and RQ4) are the performance metrics used to measure the effect of compaction on the Cassandra performance for a Docker based platform. These performance metrics are identified from the literature review, which is done to answer RQ1.

3.2.2.2 Independent variables

The independent variables of the experiments for answering RQ3 are the three different workloads and for answering RQ4 are the compaction parameters used to tune the Cassandra performance for a Docker based platform. The parameters are identified from the literature review, which is done to answer RQ2.

After conducting the experiments using these dependent and independent variables, the impact of compaction on Cassandra’s performance for a Docker based platform (using the default setting of Cassandra) is analyzed from the results of the experiments conducted using different workloads (independent variable for RQ3). This helps in answering RQ3. Additionally, the compaction parameters (independent variable for RQ4) are tuned to find the sub-optimal value of each parameter, which helps in answering RQ4. The dependent and independent variables used for answering RQ3 and RQ4 are as shown below.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Variables</th>
<th>For RQ3</th>
<th>For RQ4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dependent</td>
<td>Performance metrics</td>
<td>Performance metrics</td>
</tr>
<tr>
<td>2</td>
<td>Independent</td>
<td>Three different workloads</td>
<td>Compaction parameters</td>
</tr>
</tbody>
</table>

Table 1: Dependent and independent variables for answering RQ3 and RQ4

3.2.3 Designing the Test environment

The design of the different nodes used to perform the experiments is shown in Figure 6 below.

Figure 6: Design of the test environment
3.2.3.1 Load generator node (Node1)

Cassandra 3.0.8 is downloaded and installed by tarball installations on this node and is used mainly for the purpose of generating the different workloads to carry out the experiments on Node2. It is also used to measure the latency of each operation while performing the experiments.

Cassandra 3.0.8 package includes the Cassandra stress tool. The location of the stress tool inside the Cassandra package is cassandra_package/tools/bin folder and it needs be run from this folder. This tool is used for stress loading a node with data and also for measuring the latency of each operation, by using the stress command.

3.2.3.1.1 Stress command for write operations

This operation needs to be performed before any mixed load operation is performed. The basic stress command for performing a write operation is as follows:

```
"./cassandra-stress write n=q -node ipaddress"
```

1. `./cassandra-stress` – it enforces the stress tool to start performing a stress operation
2. `write n=q` – it performs only write operations with the database being filled with ‘n’ value
3. `-node ipaddress` – it is used to specify the ipaddress of the node onto which the stress needs to be done

3.2.3.1.2 Stress command for mixed load operations

The basic stress command used for a mixed load operation is as follows:

```
"./cassandra-stress mixed ratio(write=x,read=y) duration=t cl=ONE -pop dist=UNIFORM(1..q) -rate threads=z -log file=file_location -node ipaddress"
```

1. `./cassandra-stress` – it enforces the stress tool to start performing a stress operation
2. `mixed ratio(write=x,read=y)` – It uses a mixed workload of ‘x%’ writes and ‘y%’ reads
3. `duration=t` – is the time to perform the experiments need to be specified. ‘t’ can be specified in seconds by ‘s’, in minutes by ‘m’ and in hours by ‘h’ along with the duration value
4. `cl=one` – it is the consistency level with the default value ‘one’
5. `-pop dist=UNIFORM(1..q)` – it represents the distribution of the dataset with ‘q’ being the number of operations (i.e. n=q)
6. `-rate threads=z` – it is used to specify the number of threads used for the experiments. ‘z’ is the number of threads in this case
7. `-log file=file_location` – it is used to store the latency value in a log file, by creating and specifying the file name and file location respectively
8. `-node ipaddress` – it is used to specify the ipaddress of the node onto which the stress needs to be done

3.2.3.2 Experiment node (Node2)

The process involved in setting up the testing environment on Node2 is as follows:

1. Docker is downloaded and installed on node2. Docker image of Cassandra 3.0.8 is downloaded by using the command:
```
docker run --name some-cassandra -d -e
CASSANDRA_BROADCAST_ADDRESS="ip_address_of_your_node" -p
7000:7000 -p 9042:9042 cassandra:3.0.8"
```
a. **Docker run** command downloads the images and, then creates a
new container and starts the Cassandra:3.0.8 inside the container
b. **--name some-cassandra** is used to specify the container name
running Cassandra:3.0.8
c. **-d** is used to run Cassandra in the detached mode
d. **-e** is used to set any environment variable
e. **CASSANDRA_BROADCAST_ADDRESS="ip_address_of_your_node"** is used to broadcast the other nodes that Cassandra is
running in the ip_address provided
f. **-p 7000:7000 -p 9042:9042** are the port addresses of the
Cassandra
g. **cassandra:3.0.8** is the image to be downloaded along with the tag
or the version

2. After downloading the Docker image, we need to check if the container is
in active state, that is, whether its running or not. It is checked by using the
command:
```
docker ps -a
```
This command gives a complete list of running and exited containers. We
need to copy the container id from the list for the next step.

3. Then we need to enter the container and go to the bash location. This is
done by using the command:
```
docker exec –it containerid bash
```
4. Then we need to check whether the Cassandra is running or not. This is
Done by the command:
```
nodetool status
```
5. Then we need to check for the cassandra.yaml file, which consists all the
settings of Cassandra. It is found to be in the ‘/etc/cassandra’ folder inside
the container. Then we need to download the dependent binaries and
libraries required for our experiment. This is done by:
   a. **“apt-get update”** – updates the file system inside the container
   b. **“apt-get upgrade”** – upgrades the file system inside the container
c. **“apt-get install vim”** – installs vim editor
d. **“apt-get install jython”** – installs jython used to run and compile
   the combination of java and python file

6. Then we need to create a folder for storing the Cassandra metrics. The file
created is ‘cassmetrics’. Then we come out of the container without
exiting or stopping the container by using the keys ‘control+P+Q’.

7. On Node2, we write two scripts, one for collecting the OS metrics and the
other for collecting Cassandra metrics. OS metrics script is written in
‘python’ and is named ‘osmetrics.py’. Cassandra metrics script is written
in ‘jython’ and is named ‘cassmetrics.py’. The metrics used in the python
and the jython scripts are derived from the Literature Review.
8. We need to download ‘python’ for running the osmetrics.py file on Node2. Then two folders with the names ‘osmetrics’ and ‘dockermetrics’ are created to store the OS metrics and the Docker metrics respectively.

9. Then the Cassandra metrics scripts need to be sent to the location where the ‘cassandra.yaml’ file is present. The ‘cassandra.yaml’ file is present inside the container and in the location ‘/etc/cassandra’. This done by using the command:
   “dockercp cassmetrics.py containerid:/etc/cassandra”

10. The commands to collect the OS metrics, Docker metrics and Cassandra metrics needs to be executed where their respective files are present. The commands used respectively are as follows:
   a. “python osmetrics.py > osmetrics” – the command when executed stores the output in the folder ‘osmetrics’ created earlier
   b. “docker stats > dockerstats” – when executed stores the output in the folder ‘dockerstats’ created earlier
   c. “jython cassmetrics.py > cassmetrics” – when executed inside the container, it stores the output in the cassmetrics folder created earlier

3.2.4 Essential instructions before performing experiments

   We need to be familiar with the different instructions before conducting the experiments. The instructions are as follows:

3.2.4.1 Execution of the commands

   The command to stress is run on Node1. The commands to measure: the OS metrics is run on Node2, the Docker metrics is run from Node2 and the Cassandra metrics is run from the Docker container on Node2. These commands need to be executed simultaneously, while the experiment is conducted, to collect all the metrics for the same experiment.

3.2.4.2 Deletion of data after every experiment

   The data gets stored in to the Cassandra database, while each experiment is being carried out. This data needs to be deleted in order to perform new experiments in a clean environment.

   The data in Cassandra gets stored in the table ‘standard1’ within the ‘Keyspace1’, when a stress is done. This data needs to be deleted or truncated. This is done by using the Cassandra Query Language (CQL) commands in the CQLSH tool. This needs to be carried out inside the container running Cassandra.

   The data deleted from the tables gets stored in the ‘snapshots’ folder as a backup mechanism of Cassandra. This ‘snapshots’ folder is accessed by going to the location ‘/var/lib/Cassandra/data/keyspace1/standard1’. The data stored with the name ‘standard1’ inside the ‘snapshots’ folder need to be deleted. After this the new experiment can be started.
3.2.4.3 Bringing the changes in the yaml file into effect

1. The yaml file ‘cassandra.yaml’ inside the Docker container is edited using a text editor ‘Vim’, installed earlier. Any changes made to the yaml file is applicable only when the Cassandra process is restarted.

2. For this, we need to kill the process id of the Cassandra. To check the process id, the command used is:
   
   "ps -ef | aux cassandra"

3. After getting the process id, we need to kill the process. This is done by:
   
   "sudo kill process _id"

4. This kills the Cassandra process and stops the Docker container running Cassandra. We need to restart the container to bring the Cassandra into running state. This is done by:
   
   “docker restart containerid”

5. We can enter into the container by using the previous commands as given in the Section 3.2.3.2 and perform the experiments with the new Cassandra settings.

3.2.5 Performing the experiments

After designing the test environment and understanding the important instructions for performing the experiments, the experiments are conducted. The different experiments conducted are as follows:

3.2.5.1 Selecting number of operations (n), workloads and duration

The motivation for selecting the values are as follows:

1. The number of operations to perform the experiments are selected to be one million and duration of each experiment is chosen to be 20 minutes (20m). These values are chosen based on the values used by the author in [31] to perform the experiments, where huge amount of data is inserted and sufficient time is provided for compactions to occur.

2. The three different workloads chosen are (1) 90%writes and 10%reads, (2) 70%writes and 30%reads, and (3) 50%writes and 50%reads. In this thesis, we mainly intended to investigate the impact of compaction on Cassandra’s performance when mixed workloads with high percentage of write operations (i.e. 90%writes-10%reads and 70%writes- 30%reads) are used. The 50%writes-50%reads is a balanced workload and is chosen to investigate if the workloads with high percentage of write operations have greater impact than the balanced workload.

3.2.5.2 Experiments to find number of threads

This experiment is conducted to find out the number of threads that yield the maximum operation rate. This experiment is conducted for each of the workloads: (1) 90%writes and 10%reads, (2) 70%writes and 30%reads, and (3) 50%writes and 50%reads. The duration is not specified in the stress command, which helps in finding
the number of threads that have the maximum operation rate. The process to find the number of threads is as follows:

1. A write operation is performed at the start of each experiment, to carry out mixed load operations in the next step. The stress command is as follows:

```
cassandra-stress write n=1000000 -node 194.47.131.210
```

2. The stress command without the duration, to find the number of threads for different workloads is as follows:
   a. 90%writes and 10%reads

```
cassandra-stress mixed ratio(write=9,read=1) n=1000000 cl=ONE -pop dist=UNIFORM(1..1000000) -log file=~/logs5/90_10_10lakh.log -node 194.47.131.210
```
   b. 70%writes and 30%reads

```
cassandra-stress mixed ratio(write=7,read=3) n=1000000 cl=ONE -pop dist=UNIFORM(1..1000000) -log file=~/logs5/70_30_10lakh.log -node 194.47.131.210
```
   c. 50%writes and 50%reads

```
cassandra-stress mixed ratio(write=5,read=5) n=1000000 cl=ONE -pop dist=UNIFORM(1..1000000) -log file=~/logs5/50_50_10lakh.log -node 194.47.131.210
```

3. The results and analysis of the experiments are provided in the Chapter 4. These results help in conducting the experiments to answer research question RQ3 and RQ4.

### 3.2.5.3 Experiments with default settings of Cassandra (for answering RQ3)

This experiment is conducted to answer the research question RQ3. The experiments are conducted by using three different workloads: (1) 90%writes and 10%reads, (2) 70%writes and 30%reads, and (3) 50%writes and 50%reads. Each experiment is conducted for a duration of 20 minutes and 10 iterations are performed for validating the results of each experiment. These experiments are carried out by keeping in mind the essential instructions as described in the Section 3.2.4.1 and Section 3.2.4.2. Then average and RMSE of the metrics collected are calculated. The commands to perform the experiments with different workloads are as follows:

1. A write operation is performed at the start of each experiment, to carry out mixed load operations in the next step. The Stress command is as follows:

```
cassandra-stress write n=1000000 -node 194.47.131.210
```

2. Then the Stress command, command to measure OS metrics, command to measure Docker metrics and command to measure Cassandra metrics are run simultaneously. The commands for the different workloads are as follows:
   a. 90%writes and 10%reads workload

   i. stress Command
ii. OS metrics

“python osmetrics.py > osmetrics_90_10/90_10_1.csv”

iii. Docker metrics

“docker stats > dockerstats_90_10/90_10_1.csv”

iv. Cassandra metrics

jython cassmetrics.py > cassmetrics_90_10/90_10_1.csv

b. 70%writes and 30% reads workload

i. Stress command

“./cassandra-stress mixed ratio(write=7,read=3) duration=20m cl=ONE -pop dist=UNIFORM(1..1000000) –rate threads=406 -log file=~/70_30_logs/70_30_1.log -node 194.47.131.210”

ii. OS metrics

“python osmetrics.py > osmetrics_70_30/70_30_1.csv”

iii. Docker metrics

“docker stats > dockerstats_70_30/70_30_1.csv”

iv. Cassandra metrics

jython cassmetrics.py > cassmetrics_70_30/70_30_1.csv

c. 50%writes and 50%reads workload

i. Stress command

“./cassandra-stress mixed ratio(write=5,read=5) duration=20m cl=ONE -pop dist=UNIFORM(1..1000000) –rate threads=609 -log file=~/50_50_logs/50_50_1.log -node 194.47.131.210”

ii. OS metrics

“python osmetrics.py > osmetrics_50_50/50_50_1.csv”

iii. Docker metrics

“docker stats > dockerstats_50_50/50_50_1.csv”

iv. Cassandra Metrics
3. After conducting the experiments, the results are collected and then analysed which are provided in the Section 4.2.

3.2.5.4 Experiments after tuning the parameters (for answering RQ4)

These experiments are conducted for only the write heavy workloads (i.e. 90%writes and 10%reads). These experiments are conducted by changing the parameters in the yaml file, which are related to the compaction. This experiment is carried out to answer the research question RQ4. Each experiment is conducted for a duration of 20 minutes and 10 iterations are performed for validating the results of each experiment. These experiments are carried out by keeping in mind the essential instructions as described in the Section 3.2.4. The experimental results are used to calculate average and RMSE (Root Mean Square Error) values. KPIs are selected to compare these average and RMSE values with the average and RMSE values of the default parameters (calculated in RQ3). The results are provided in Section 4.3. The different parameters that can be tuned and if these parameters have an impact of compaction on Cassandra’s performance are shown in the table below.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Compaction parameters</th>
<th>Impact on Cassandra’s performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Memtable_heap_space_in_mb</td>
<td>Has impact</td>
</tr>
<tr>
<td>2</td>
<td>Memtable_offheap_space_in_mb</td>
<td>Has impact</td>
</tr>
<tr>
<td>3</td>
<td>Memtable_flush_writers</td>
<td>Has impact</td>
</tr>
<tr>
<td>4</td>
<td>Memtable_cleanup_threshold</td>
<td>Has impact</td>
</tr>
<tr>
<td>5</td>
<td>Concurrent_compactors</td>
<td>Has impact</td>
</tr>
<tr>
<td>6</td>
<td>Compaction_throughput_mb_per_sec</td>
<td>Has impact</td>
</tr>
<tr>
<td>7</td>
<td>Compaction_largepartition_threshold_mb_per_sec</td>
<td>Has no impact</td>
</tr>
<tr>
<td>8</td>
<td>Snapshot_before_compaction</td>
<td>Has no impact</td>
</tr>
</tbody>
</table>

3.2.5.4.1 Memtable_heap_space_in_mb

1. The default value of Memtable_heap_space_in_mb is ‘2048’ (which is the on-heap memory) and it is 1/4th the size of the heap. Also, Memtable_offheap_space_in_mb (which is off-heap memory) is 1/4th size of the heap. And, memTable_cleanup_threshold uses the sum of Memtable_heap_space_in_mb and Memtable_offheap_space_in_mb to flush data into SSTables. In this thesis, we prefer to only change the on-heap memory over off-heap memory to investigate the impact of compaction on Cassandra’s performance.

2. Five different experiments using different heap values are conducted to check the impact of this parameter on the performance of Cassandra, running inside a Docker container. First, three different values of heap are chosen (i.e. 4096, 6144 and 8192), which are multiples of 2048 until we reach the heap size. Then the values 3072 and 5120 are chosen to plot more evenly distributed points on the graph and investigate the impact of compaction.

3. The different values of heap considered for the experiments are:
   a. Memtable_heap_space_in_mb = 3072
b. Memtable_heap_space_in_mb = 4096

c. Memtable_heap_space_in_mb = 5120

d. Memtable_heap_space_in_mb = 6144

e. Memtable_heap_space_in_mb = 8192

4. These values are changed one at a time and other parameter values are kept as default.

5. Similar commands for stress, OS metrics, Docker metrics and Cassandra metrics are used to measure the metrics, which were used in the previous experiment in the Section 3.2.5.3.

3.2.5.4.2 Memtable_flush_writers and memTable_cleanup_threshold

1. The default value of Memtable_flush_writers is 8 and memTable_cleanup_threshold is ‘0.11’. memTable_cleanup_threshold is calculated by the formula:

   "memTable_cleanup_threshold = 1/(1 + Memtable_flush_writers)"

2. Three different experiments are conducted by changing the values of these parameters in multiples of 2 to check of the effects of these parameters on the Cassandra’s performance running inside a Docker container. As per the documentation of Datastax [32], more the value of memTable_cleanup_threshold, less is the flush activity. So, the value 0.14 (i.e. greater than default which is 0.11) is chosen. The values 0.09 and 0.08 (i.e. lesser than default) are chosen to investigate if there is any impact of compaction when more flush activity is done.

3. The different values considered for conducting experiments are:
   a. memtable_flush_writers = 6 & memTable_cleanup_threshold = 0.14
   b. memtable_flush_writers = 10 & memTable_cleanup_threshold = 0.09
   c. memtable_flush_writers = 12 & memTable_cleanup_threshold = 0.08

4. These values are changed one at a time and other parameter values are kept as default.

5. Similar commands for stress, OS metrics, Docker metrics and Cassandra metrics are used to measure the metrics, which were used in the previous experiment in the Section 3.2.5.3.

3.2.5.4.3 Concurrent_compactors

1. The default value of this parameter is 1.
2. Three different experiments are conducted using three different concurrent_compactors values to check if there is any effect on the performance of Cassandra running inside a Docker container. As per the documentation of Datasatx [32], it is recommended to increase the compactors value to the number of core present in the system being used. In our testing environment, we have 12 cores and each core has 2 threads. So, the values 12 and 24 are chosen. The value 18 is chosen to plot more evenly distributed points in the graph showing the impact of compaction.

3. The different values considered for conducting experiments are:
   a. Concurrent_compactors = 12
   b. Concurrent_compactors = 18
   c. Concurrent_compactors = 24

4. These values are changed one at a time and other parameter values are kept as default.

5. Similar commands for stress, OS metrics, Docker metrics and Cassandra metrics are used to measure the metrics, which were used in the previous experiment in the Section 3.2.5.3.

3.2.5.4.4 Compaction_throughput_mb_per_sec

1. This parameter has default value as 16.

2. Three different experiments are carried out to find the impact of this parameter on Cassandra's performance running inside a Docker container. As per the documentation of Datasatx [32], the recommended values of throughput are 16 to 32 times. So, 24 and 32 are chosen. 40 is chosen to investigate if there is any impact of compaction and if there is any deviation from the recommended values.

3. The different values used for the experiments are:
   a. Compaction_throughput_mb_per_sec = 24
   b. Compaction_throughput_mb_per_sec = 32
   c. Compaction_throughput_mb_per_sec = 40

4. These values are changed one at a time and other parameter values are kept as default.

5. Similar commands for stress, OS metrics, Docker metrics and Cassandra metrics are used to measure the metrics, which were used in the previous experiment in the Section 3.2.5.3.
3.2.5.5 Experiment with the sub-optimal parameters (when used together) found in RQ4

This experiment is carried out with 90% Writes and 10% reads workload using the sub-optimal parameters found using the experiments performed in Section 3.2.5.4.1, Section 3.2.5.4.2, Section 3.2.5.4.3, and Section 3.2.5.4.4. The cassandra.yaml file is updated with the sub-optimal values found earlier in Section 3.2.5.4 (found in RQ4) to investigate the impact of compaction on Cassandra’s performance running inside Docker. The experiment is carried out for 20 minutes and is repeated 10 times (i.e. 10 iterations are performed) to validate the results. Then the average and RMSE of the KPIs (same KPIs used in Section 3.2.5.4) are calculated. This KPIs values are compared with the KPIs values of the default parameters (calculated in RQ3).
4 RESULTS AND ANALYSIS

This chapter provides the description and analysis about the outcome of the different experiments performed. The outcome and the analysis of the different experiments performed are as follows:

4.1 Results for number of threads experiment

The experiment conducted to find the number of threads is described in the Section 3.2.5.2. The stress command is used without specifying the duration of the experiment. So, the stress command uses the default value of the threads to perform the experiment and it goes on increasing the number of threads, until the experiment achieves the maximum operation rate. After achieving the maximum operation rate, the experiment stops automatically. This gives us the number of threads required for the experiment for different workloads. The number of threads for the different workloads used are as shown in the Table 3.

<table>
<thead>
<tr>
<th>S.no</th>
<th>Workloads</th>
<th>Number of threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>90%Writes and 10%Reads</td>
<td>609</td>
</tr>
<tr>
<td>2</td>
<td>70%Writes and 30%Reads</td>
<td>406</td>
</tr>
<tr>
<td>3</td>
<td>50%Writes and 50%Reads</td>
<td>609</td>
</tr>
</tbody>
</table>

4.2 Results and analysis for answering RQ3

All the Cassandra and OS metrics for different workloads are calculated through these experiments and are analysed. Out of these, some important metrics are chosen which provide with a great detail of the impact of Cassandra compaction (with the default Cassandra setting) on its performance running inside Docker container. Four out of the five compaction metrics are chosen and compaction rate metric is left out because compactions in Cassandra do not occur per second rather occur when similar sized SSTables are merged into a single SSTable. Four out of thirteen OS metrics are chosen to represent the results. These OS metrics are chosen based on the work done by the authors in [33]. The authors in [33], use CPU utilization, Disk utilization and Memory utilization for evaluating the performance of container-based OS virtualization. Our thesis uses these OS metric along with the OS metric ‘Latency mean’ to investigate the impact of compaction. The different compaction metrics and OS metrics chosen to represent the results are as shown below.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Compaction metrics</th>
<th>OS metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total compaction count</td>
<td>Disk usage percentage</td>
</tr>
<tr>
<td>2</td>
<td>Total bytes compacted</td>
<td>CPU usage percentage</td>
</tr>
<tr>
<td>3</td>
<td>Completed tasks</td>
<td>Memory usage percentage</td>
</tr>
<tr>
<td>4</td>
<td>Pending tasks</td>
<td>Latency mean</td>
</tr>
</tbody>
</table>
4.2.1 Results and analysis of compaction metrics

1. Total compaction count:

The compaction count is high in the 90% Writes-10% Reads workload when compared to the other workloads. This shows that the number of writes are directly proportional to the number of compactions occurring. It is seen that there were around 17 compactions on an average in the 90% Writes-10% Reads workload and around 15 compactions on an average in 70% Writes-30% Reads workload and around 8 compactions on an average in 50% Writes-50% Reads workload.

Also, in each iteration for a workload the total number of compactions is not a constant value and keeps changing. This is because of the amount of data compacted varies in each case. This shows that the number of compactions can be varying and cannot be predicted accurately, but a range can be predicted.

![Figure 7: Total number of compactions for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads](image)

2. Total bytes compacted:

As seen in the total compaction count the bytes compacted per second also increase in the same way i.e. more bytes compacted with more writes heavy workload. This implies that the bytes compacted would be more for 90% Writes-10% Reads workload and keeps on decreasing continuously for 70% Writes-30% Reads and 50% Writes-50% Reads workloads. This is because with the increase in the number of compactions the bytes compacted also increases.

Also, in each iteration for a workload the total number of bytes compacted is not a constant value and keeps changing because of the number of completed and pending tasks. This shows that the number of bytes can be varying.
3. **Completed tasks:**

Completed task here refers to a Completed compaction task. The number of compaction tasks that are completed are similar in both 90%Writes-10%Reads and 70%Writes-30%Reads workloads. Since the number of compactions are similar in both the workload, the completed tasks are also in the similar range for these workloads. Also it can be seen that in 50%Writes-50%Reads workload, the completed tasks are less when compared to their counterparts as the number of compactions performed in this workload are relatively less.

4. **Pending tasks:**

Pending tasks are Pending compaction tasks that are yet to be performed or that won’t be performed by Cassandra. The pending tasks are comparatively quite low when compared to the completed tasks. The pending tasks are almost in the same range for all the three different workloads as seen in the graphs below. This shows that irrespective of the workload there can be the similar number of pending compaction tasks.
Figure 10: Total number of pending tasks for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads

5. Average and RMSE value of the compaction metrics for different workloads:

The average value and RMSE value of the Cassandra compaction metrics collected for the different workloads are as follows:

Table 5: Average and RMSE values of compaction metrics for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads

<table>
<thead>
<tr>
<th>S.No</th>
<th>Compaction Metric</th>
<th>Average ± RMSE for 90-10 workload</th>
<th>Average ± RMSE for 70-30 workload</th>
<th>Average ± RMSE for 50-50 workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Total compaction count</td>
<td>16.6 ± 0.05777778</td>
<td>14.8 ± 0.05333333</td>
<td>8.5 ± 0.06944444</td>
</tr>
<tr>
<td>2</td>
<td>Total Bytes compacted (in GB)</td>
<td>16.53869098 ± 0.9711533</td>
<td>10.1962026 ± 0.696552</td>
<td>4.479496924 ± 0.1730866</td>
</tr>
<tr>
<td>3</td>
<td>Completed tasks</td>
<td>516.3 ± 68.300556</td>
<td>526.3 ± 40.0227778</td>
<td>393.1 ± 65.2938889</td>
</tr>
<tr>
<td>4</td>
<td>Pending tasks</td>
<td>3.3 ± 0.01166667</td>
<td>2.7 ± 0.02277778</td>
<td>3.6 ± 0.01333333</td>
</tr>
</tbody>
</table>

4.2.2 Results and analysis of OS metrics

1. Disk usage:

Along with the Cassandra metrics, few OS metrics are also calculated in this experiment. Disk usage percentage is one of them. The disk usage percentage is calculated at every instance of the experiment and it is seen how it varies in this time. A graph is plotted for the same and is shown in Figure 11. It is clear that the disk usage is similar irrespective of the workload. Also, the graph clearly shows that in few places there is a steep fall in the disk usage. This is because, the stress tool for mixed load, writes some data and deletes it eventually when the stress is finished. All the rows which are to be deleted are marked as Tombstones by Cassandra and are removed during compaction, which means that the data is present in Cassandra until compaction occurs (Once a row is marked with Tombstone the user cannot use it). This means that at one point there would be two instances of the same data i.e. not compacted data and the compacted data. When a compaction occurs, the
tombstones are removed and free space is formed which is the reason for steep fall in disk usage. This is the reason for the spikes in the data usage.

The Disk usage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%read workloads are shown in the Figure 11.

![Figure 11: Disk usage percentage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads](image)

2. **CPU usage:**

The graph in Figure 12 shows that in 90%Write-10%Read and 70%Write-30%Read workloads, the majority of CPU usage lies between the range 90% to 100%. From this it can be inferred that, for write heavy loads the CPU usage always lie on the higher bounds because of the increase in compactions and data. In contrast to that, the 50%Writes-50%Reads workload shows most of the CPU usage values below 90%. There are few spikes in all the graphs which may be because of low load on the CPU i.e. no running compactions or other Cassandra related operations or Server I/O operations or some other processes running in the root.

![Figure 12: CPU usage percentage for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads](image)

3. **Memory usage:**

Memory usage has been similar in all the three workloads. This shows that when using Docker, the RAM usage is always in the same range irrespective of the workload. This is a benefit in the work heavy workload as there might be more data to be written on Cassandra but this won’t affect the memory usage greatly. [15]
4. **Latency mean:**

   Latency mean here refers to the average time taken for a particular task to complete. This is measured in milliseconds. From the graphs it is seen that the 90% Writes-10% Reads and 70% Writes-30% Reads workloads have similar latencies when compared to the 50% Writes-50% Reads workload. In 50% Writes-50% Reads workload the latencies are almost twice higher when compared to their counterparts. As the number of compactions increase, the latency mean decrease. This is because the SSTables in which the data is to be searched are decreased and the reason behind it is that the SSTables are merged while a compaction process occurs to form a single and large SSTable. This is the reason why the latency mean is high in 50% Writes-50% Reads workload when compared to its counterparts.

5. **Average value and range of the OS metrics:**

   The average and RMSE values of the OS metrics collected for the different workloads are as follows:
Table 6: Average and RMSE values of OS metrics for 90%Write-10%Read, 70%Write-30%Read and 50%Write-50%Read workloads

<table>
<thead>
<tr>
<th>S.No</th>
<th>OS Metric</th>
<th>Average ± RMSE for 90-10 workload</th>
<th>Average ± RMSE for 70-30 workload</th>
<th>Average ± RMSE for 50-50 workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Disk usage</td>
<td>10.0994 ± 0.04003241</td>
<td>9.93425 ± 0.01165785</td>
<td>9.536275 ± 0.03109169</td>
</tr>
<tr>
<td>2</td>
<td>CPU usage</td>
<td>92.53907423 ± 0.00986247</td>
<td>92.08053334 ± 0.07230821</td>
<td>88.48474167 ± 0.00459532</td>
</tr>
<tr>
<td>3</td>
<td>Memory usage</td>
<td>34.33904167 ± 0.00815263</td>
<td>34.09646667 ± 0.00121045</td>
<td>33.52303333 ± 0.00447461</td>
</tr>
<tr>
<td>4</td>
<td>Latency</td>
<td>7.9 ± 0.00044444</td>
<td>7.19 ± 0.00716111</td>
<td>13.94 ± 0.00035556</td>
</tr>
</tbody>
</table>

### 4.3 Results and analysis for answering RQ4

The experiments performed for tuning the compaction parameters is done as described in the Section 3.2.5.4. It is performed using the same number of operations, that is ‘one million’ and the same number of threads, that is ‘609’ for the write heavy workload. The experiment are conducted by using the 90%Writes-10%Reads workload. The parameters are changed by accessing the cassandra.yaml file. Each parameter is changed separately to find the impact on Cassandra compaction and then the obtained results are compared with the results of the Section 4.2 (default Cassandra setting results).

Write heavy workload inserts data at a very quick pace. This data needs to be stored into the disks at a very quick pace. Compaction plays a vital role in storing data into disks. So, in order to store the huge amount of data into disks, more amount of data needs to be compacted and in order to compact more data, more number of compactions need to occur. This can be calculated by using ‘Total compaction count’ and ‘Bytes compacted’ metrics. Also the author in [31] suggests more the amount of data compacted, better is the performance of Cassandra and for compacting more data, more compactions are required. So, a base is required to compare the experimental results of the tuned parameters to the results of the default parameters. So, KPIs are selected as a base to compare the experimental results. The KPI selected are: ‘Total compaction count’ and ‘Bytes compacted’ metrics. These KPIs are selected because they provide us with the number of the compactions that has occurred and also the amount of data compacted.

We then calculate the RMSE (Root Mean Square Error) for evaluating the results of the experiment [34][35]. The RMSE gives us the error rate. This error rate helps in calculating the error range of the dependent variable. This error helps in understanding the variation of the dependent variable from its average value. The average and RMSE value of the KPIs (collected from the experiments after changing the parameter) are compared with the average and RMSE value of the KPIs for default parameters. The results obtained by changing the different parameters are graphical representation of the metrics (KPI) vs value of the parameter. The graph shows the average value and also the error (RMSE) for each of the metrics (KPI). The different parameters tuned in the thesis are as follows:

#### 4.3.1 Memtable_heap_space_in_mb

The memtable_heap_space_in_mb is varied in the cassandra.yaml file and the results have been depicted in the graphs below:
From the graphs it can be seen that the bytes compacted and compaction rate have higher values when the Memtable_heap_space_in_mb is 5120. In this case the graph shows a peak at 5120 in both bytes compacted and compaction count. This implies that Cassandra’s performance is high at this value and gradually decreases from here as seen in the graph. So, 5120 is considered a sub-optimal value in the case of memtable_heap_space_in_mb.

The average and RMSE values for different value of Memtable_heap_space_in_mb is shown in the table below.
Table 7: Average and RMSE values of Memtable_heap_space_in_mb for bytes compacted and compaction count

<table>
<thead>
<tr>
<th>S.No</th>
<th>Memtable_heap_space_in_mb value</th>
<th>Average ± RMSE for bytes compacted (in GB)</th>
<th>Average ± RMSE for compaction count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default (2048)</td>
<td>16.538691 ± 0.9711533</td>
<td>16.6 ± 0.05777778</td>
</tr>
<tr>
<td>2</td>
<td>3072</td>
<td>11.5510286 ± 0.5237426</td>
<td>14.3333333 ± 0.89</td>
</tr>
<tr>
<td>3</td>
<td>4096</td>
<td>16.1696056 ± 0.5675639</td>
<td>7.3333333 ± 0.06</td>
</tr>
<tr>
<td>4</td>
<td>5120</td>
<td>65.4305837 ± 0.21193422</td>
<td>19 ± 0</td>
</tr>
<tr>
<td>5</td>
<td>6144</td>
<td>34.889781 ± 0.21122386</td>
<td>4 ± 0</td>
</tr>
<tr>
<td>6</td>
<td>8192</td>
<td>16.490811 ± 0.52127</td>
<td>2 ± 0</td>
</tr>
</tbody>
</table>

4.3.2 Memtable_flush_writers and memTable_cleanup_threshold

The Memtable_flush_writers are varied in the cassandra.yaml file and the results have been depicted in the graphs below:

![Figure 17: Bytes compacted for Memtable_flush_writers](image-url)
From the graphs it can be seen that the bytes compacted and compaction rate have better results when the Memtable_flush_writers is 6. But as per the documentation of Datastax [32], less the flush writers, slower is the flush activity and in turn less is the number of compactions. But the value of total compaction count are similar when Memtable_flush_writers is set to 8(default) and 6. And, from the graph it can be seen that the bytes compacted value for 6 is more than the default. This shows that the value 6 performs better than the default and also from the other values of flush writers. So, 6 is considered to be a sub-optimum value in this case.

The average and RMSE values for different value of Memtable_flush_writers is shown in the table below.

<table>
<thead>
<tr>
<th>S.No</th>
<th>Memtable_flush_writers value</th>
<th>Average ± RMSE for bytes compacted (in GB)</th>
<th>Average ± RMSE for Total compaction count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>58.65990128 ± 0.4272617</td>
<td>16.71 ± 0.079</td>
</tr>
<tr>
<td>2</td>
<td>Default(8)</td>
<td>16.53869098 ± 0.9711533</td>
<td>16.6 ± 0.05777778</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>8.91938069 ± 0.4752822</td>
<td>12 ± 1.1666667</td>
</tr>
<tr>
<td>4</td>
<td>12</td>
<td>7.293570802 ± 0.71995406</td>
<td>9.66666667 ± 0.06</td>
</tr>
</tbody>
</table>

### 4.3.3 Concurrent_compactors

The Concurrent_compactors are varied in the cassandra.yaml file and the results have been depicted in the graphs below.
From the graphs it can be seen that the bytes compacted and compaction rate have higher values when the concurrent_compactors is 24. This is because, the bytes compacted are high and so are the compaction count which would improve the reads latency and thus making it a sub-optimum value in this case. The average and RMSE values for different value of Concurrent_compactors is shown in the table below.

**Table 9:** Average and RMSE values of Concurrent_compactors for bytes compacted and compaction count

<table>
<thead>
<tr>
<th>S.No</th>
<th>Concurrent_compactors value</th>
<th>Average ± RMSE for bytes compacted (in GB)</th>
<th>Average ± RMSE for Total compaction count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default(1)</td>
<td>16.53869098 ± 0.9711533</td>
<td>16.6 ± 0.05777778</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>169.6516581 ± 0.836</td>
<td>23 ± 0</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>219.6169487 ± 0.8549725</td>
<td>26 ± 0</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>356.1471456 ± 0.14740547</td>
<td>29 ± 0</td>
</tr>
</tbody>
</table>
4.3.4 Compaction_throughput_mb_per_sec

The Compaction_throughput_mb_per_sec are varied in the cassandra.yaml file and the results have been depicted in the graphs below:

![Figure 21: Bytes compacted for Compaction_throughput_mb_per_sec](image1)

![Figure 22: Compaction count for Compaction_throughput_mb_per_sec](image2)

From the graphs it can be seen that the bytes compacted and compaction rate have higher values when the compaction_throughput_mb_per_sec value is 16 (default). There is a significant difference in the bytes compacted but the number of compactions almost remains the same. This makes 16 the sub-optimum value for compaction_throughput_mb_per_sec.

The average and RMSE values for different value of compaction_throughput_mb_per_sec is shown in the table below.
Table 10: Average and RMSE values of Compaction_throughput_mb_per_sec for bytes compacted and compaction count

<table>
<thead>
<tr>
<th>S.No</th>
<th>Compaction_throughput_mb_per_sec value</th>
<th>Average ± RMSE for bytes compacted (in GB)</th>
<th>Average ± RMSE for Total compaction count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default(16)</td>
<td>16.53869098 ± 0.9711533</td>
<td>16.6 ± 0.058</td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td>14.58914467 ± 0.14132721</td>
<td>16.5 ± 0.125</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>14.91810926 ± 0.17424041</td>
<td>15.5 ± 0.125</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>15.70084114 ± 0.38902888</td>
<td>15.7 ± 0.1</td>
</tr>
</tbody>
</table>

4.4 Results and analysis of the experiment with the sub-optimal parameters (when used together)

The sub-optimal value of the parameters found in answering RQ4 is as shown in the table below.

Table 11: Sub-optimal value of each compaction parameter

<table>
<thead>
<tr>
<th>S.No</th>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Sub-Optimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Memtable_heap_size_in_mb</td>
<td>2048</td>
<td>5120</td>
</tr>
<tr>
<td>2</td>
<td>Memtable_flush_writers</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>concurrent_compactors</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>compaction_throughput_mb_per_sec</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

These sub-optimal values are selected and changed in the cassandra.yaml file. Experiments using these sub-optimal parameters together are conducted with 90% Writes and 10% Reads workload. KPIs are selected as Total bytes compacted and Total compaction count (similarly as selected in Section 4.3). Average and RMSE values for the selected KPIs are calculated. These KPI values are compared with the KPI values of the default parameters (collected from experiments with default parameters). The comparison of the results between the default parameters and sub-optimal parameters (when used together) is represented with graphs as shown below.
Figure 23: Comparison of bytes compacted for sub-optimal parameters (when used together) and default parameters

Figure 24: Comparison of compaction count for sub-optimal parameters (when used together) and default parameters

From the graphs it can be seen that the Total bytes compacted and the Total compaction count for the default parameters are better than the sub-optimal parameters used together. It was expected that the sub-optimal parameters used together will perform better than the default parameters. But from the results it is just the opposite and we can infer that there is some dependency between the compaction parameters and we cannot expect the sub-optimal parameters when used together to perform better than the default parameters. The average and RMSE values for sub-optimal parameters (when used together) and default parameters are shown in the table below.

Table 12: Average and RMSE values of sub-optimal parameters (when used together) and default parameters for bytes compacted and compaction count

<table>
<thead>
<tr>
<th>S.No</th>
<th>Parameters</th>
<th>Average ± RMSE for bytes compacted</th>
<th>Average ± RMSE for Total compaction count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default</td>
<td>$16.538691 ± 0.9711533$</td>
<td>$16.6 ± 0.05777778$</td>
</tr>
<tr>
<td>2</td>
<td>Sub-optimal parameters used together</td>
<td>$8.89266256 ± 0.2917$</td>
<td>$12 ± 0.293$</td>
</tr>
</tbody>
</table>
5 DISCUSSION

This chapter provides description about the Threats to Validity, Limitations of the research and the answers to the different research questions. The discussion is as follows:

5.1 Validity Threats

The different aspects of threats to validity used in the research are as follows:

5.1.1 Internal Validity

Internal validity refers to the extent to which the errors in an experiment are minimized. This indicates how well an experiment is performed and there is no effect of the confounding variables on the relationship between independent and dependent variables. [36]

The Internal validity to threats in our research is minimized by conducting the experiments for ten times, that is, ten iterations. Then average and RMSE values for the experiments are calculated. This gives us an idea how the values of the dependent variable for the 10 iterations vary with respect to the average.

5.1.2 External Validity

External validity refers to the extent to which results of the research can be generalized. [36]

The experiments conducted are designed based on the available testing environment. Accordingly, the number of operations to be performed are chosen and experiments are conducted using different workloads. The results of the experiments cannot be generalized to other testing environments, to different number of operation and to different workloads.

5.2 Limitations

There are some limitations to the scope of the thesis. The scope of the thesis is restricted to analyze the impact of Cassandra compaction on Cassandra’s performance when used inside a Docker container using Size Tiered Compaction Strategy (STCS) on a single node or a single machine. Experiments are conducted using different workloads and the output is a graphical representation of OS metrics (CPU usage, Disk usage etc.) vs time and Cassandra metrics (Bytes compacted, completed tasks etc.) vs workloads to better analyze the impact of compaction.

Also the scope of the thesis is restricted to tune the compaction parameters and analyze the sub-optimal setting of Cassandra compaction only for a write heavy workload (i.e. 90% writes-10% reads), when used on a Docker based platform. Output yielded from the experiments are represented in the form of graphs to better analyze the results and find sub-optimal value of compaction parameters for a write heavy workload, when used on a Docker based platform.
Also, the experiments performed are done using a single node cluster instead of using cluster with many nodes.

Also, the number of operations used is restricted to ‘one million’ and the different workloads used to perform the experiments are 90% Writes-10% Reads, 50% Writes-50% Reads, and 70% Writes-30% Reads.

Also, experiments for the compaction parameter MemTable_offheap_space_in_mb is not performed due to time constraints.

Also, experiments with the combination of different tuned parameters are not performed due to time constraints.

All these limitations restrict an user to be benefitted from the overall results of the experiments, when different testing environment, different workloads and different number of operations are considered.

5.3 Answers to the Research questions

RQ1: What are the performance metrics that better adhere to measure the effect of compaction on the Cassandra performance for a Docker based platform?

Performance metrics here refer to the dependent variables. These metrics are found using the Literature review. There are different Cassandra compaction metrics and OS metrics as given in the Section 2.2.1 and Section 2.2.2, that help to study the impact of compaction on the Cassandra performance for a Docker based platform. But few of these metrics are considered while representing the results. The metrics considered to represent the results are: compaction metrics (Total compaction count, Total bytes compacted, Completed tasks and Pending tasks) and OS metrics (Disk usage, CPU usage, Memory used and Latency mean).

RQ2: What are the Cassandra compaction parameters that allow to tune the Cassandra performance for a Docker based platform?

Cassandra parameters here refer to the independent variables. The cassandra.yaml file needs to be accessed in order to tune the parameters. This yaml file contains all the Cassandra parameters, of which some parameters directly affect compaction process and some parameters indirectly affect the compaction process and some parameters have no effect on the compaction process. This parameters are given in the Section 2.2.3. The parameters that directly or indirectly affect the compaction process, have an impact on the Cassandra performance. These parameters are: Memtable heap space in mb, Compaction throughput mb per second, Memtable flush writers, Memtable cleanup threshold and Concurrent compactors.

RQ3: What is the impact of Cassandra compaction (using the default settings of Cassandra compaction) on the Cassandra performance for a Docker based platform, when different workloads (1) 90%writes and 10%reads, (2) 50%writes and 50%reads, and (3) 70%writes and 30%reads are considered?

These experiments are carried out using the process mentioned in the Section 3.2.5.2 and Section 3.2.5.3. These experiments use the default Cassandra settings. The performance metrics are collected by using the python script and jython script for OS metrics and compaction metrics respectively. The results of these metrics are presented as a graphical representation of OS metrics (CPU usage, Disk usage etc.)
vs time and Cassandra metrics (Bytes compacted, completed tasks etc.) vs workloads to better analyze the impact of compaction on the Cassandra performance for a Docker based platform. The impact of compaction on Cassandra’s performance inside Docker is as follows:

1. The Total compaction count or number of compactions are directly linked to the percentage of writes. As more data is inserted into the SSTables, more frequently the data needs to be compacted in order to store it in the disks. This is the reason the total compaction count decreases from 90%Writes-10%Reads workload to 50%writes-50%reads workload. But the variation in the total compaction count is very less for 90%Writes-10%Reads and 70%Writes-30%Reads workloads.

2. As the total compaction count increases, the bytes compacted also increases as more data is compacted in 90%Writes-10%Reads and gradually decreases as the writes percentage in the workload decreases.

3. Completed compaction tasks are similar for 90%Writes-10%Reads and 70%Writes-30%Reads workloads, but for the 50%writes-50%reads workload its relatively less. This is because of the number of compactions being performed.

4. Docker containers share the same kernel of the operating system and use very less RAM while running an application. This is why the Memory usage percentage is similar for all the three workloads.

5. As Cassandra runs inside Docker, the CPU usage percentages for the different workloads are in the same range. But there are many spikes in 50%writes-50%reads workload as fewer tasks are performed when compared to 90%Writes-10%Reads and 70%Writes-30%Reads workloads.

6. The Disk usage percentage for all the workloads are similar. But, there is steep fall at some points. This is because, the write operation in the mixed workload performs frequent updates and marks some data as Tombstones. These Tombstones get deleted while a compaction occurs and disk space is freed.

7. The latency for 50%Writes-50%Reads workload is very high when compared to 90%Writes-10%Reads and 70%Writes-30%Reads workloads. This is because of the time required to read the data from the SSTables. As the number of compaction for 50%writes-50%reads workload is very less compared to the 90%Writes-10%Reads and 70%Writes-30%Reads workloads. Workloads with more write percentage have similar latency mean as they have similar number of compactions.

RQ4: What is the impact of Cassandra compaction (by tuning each compaction parameter) on the Cassandra performance for a Docker based platform, when 90%writes and 10%reads workload is considered?

The parameters that may have an impact on the compaction process are selected from the different parameters list and each parameter is tuned to different values. The experiments are carried out using the process mentioned in the Section 3.2.5.4. Performance metrics are collected and are compared with the performance metrics collected using the default Cassandra settings. Disk usage, CPU usage and Memory usage remains to be similar in all the cases and it is similar to the results obtained by using the default Cassandra settings. So, Total compaction count and Bytes compacted metrics are chosen (chosen as KPIs) for comparing the results. The average and RMSE value for each KPI is calculated, which gives a error range for that metric. These average and the error (RMSE) are compared to select an sub-optimal value in each case of tuning the parameters. In this case the parameters that
were chosen are Memtable_heap_size_in_mb, concurrent_compactors, compaction_throughput_mb_per_sec, Memtable_flush_writers and Memtable_cleanup_threshold. Based on the experiments conducted the sub-optimum values obtained for the above mentioned parameters are shown in the Table 13.

Table 13: Sub-optimal value of each compaction parameter

<table>
<thead>
<tr>
<th>S.no</th>
<th>Parameter Name</th>
<th>Default Value</th>
<th>Sub-Optimal Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Memtable_heap_size_in_mb</td>
<td>2048</td>
<td>5120</td>
</tr>
<tr>
<td>2</td>
<td>Memtable_flush_writers</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>concurrent_compactors</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>compaction_throughput_mb_per_sec</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

Next, an additional experiment is conducted using the sub-optimal parameters together with 90% Writes and 10% Reads workload as described in the Section 3.2.5.5. This is done to compare the performance of sub-optimal parameters used together and the default parameters. The results show that the default parameters perform better than the sub-optimal parameters used together. This shows there is some dependency between the compaction parameters, and sub-optimal value of a compaction parameter may not be suitable with sub-optimal value of another compaction parameter.


6 CONCLUSION AND FUTURE WORK

The Conclusion and Future work are described in this chapter. They are as follows:

6.1 Conclusion

The main aim of the thesis was to analyze the impact of Cassandra compaction on Cassandra’s performance used inside Docker container.

The research starts with a Literature review to find the different performance metrics to analyze the performance of Cassandra and different compaction parameters that can be tuned to have an impact on Cassandra’s performance. These metrics are used to collect the results of the experiments conducted.

The initial experiments were conducted using the default settings of Cassandra with one million operations and different workloads (90% Writes and 10% Reads, 50% Writes and 50% Reads, and 70% Writes and 30% Reads). The dependent variables (performance metrics) are collected by changing the independent variables (different workloads) and the result is presented as a graphical representation of OS metrics (CPU usage, Disk usage etc.) vs time and Cassandra metrics (Bytes compacted, completed tasks etc.) vs workloads to better analyze the impact of compaction on the Cassandra performance for a Docker based platform. From the graphs, it is clear that the Memory usage and Disk usage for the different workloads are similar. But the CPU usage and Latency for 50% Writes-50% Reads workload varies from the counterparts as fewer tasks are performed and it takes more time to complete a read request because of less number of compaction. The Total compaction count, bytes compacted and completed tasks depend directly on the percentage of writes and they continuously decrease from 90% Writes-10% Reads workload to 50% Writes-50% Reads workload.

Then the parameters are tuned and experiments are conducted using ‘one million’ operations and write heavy workload (90% Writes and 10% Reads). The same dependent variables (performance metrics) are collected by changing the independent variables (compaction parameters). These collected metric results are compared with the results of the performance metrics collected by using the default Cassandra setting. This comparison is presented as a graphical representation showing the average value and RMSE value of the performance metric vs tuned parameter value to better analyze the impact of compaction when a particular parameter is tuned.

Based on the experiment conducted above, the sub-optimum values for few cassandra parameters are proposed i.e. for Memtable_heap_size_in_mb it is 5120, for concurrent_compactors it is 24, for compaction_throughput_mb_per_sec it is 16, for Memtable_flush_writers it is 6 and for Memtable_cleanup_threshold it is 0.14. These parameters are calculated by analyzing the bytes compacted and compaction count metrics using RMSE. The sub-optimum values of the above mentioned parameters are chosen to form an sub-optimal setting of Cassandra and an additional experiment is conducted to investigate if it performs better than the default setting of Cassandra on a Docker-based platform. But the results shows that default setting performs better than the sub-optimal setting. This infers that there is some dependency between the compaction parameters and sub-optimal parameters (when used together) do not always perform better.
6.2 Future work

We have examined the impact of Cassandra compaction on the Cassandra’s performance inside Docker. We have carried out our research to a certain extent and more research needs to be done in the field of ‘Cassandra running inside Docker container’.

Default Cassandra setting uses Size Tiered Compaction Strategy (STCS) as the default compaction strategy. In the future, experiments can be conducted using the compaction strategies (Leveled Compaction Strategy and Date Tiered Compaction Strategy).

Also, a single node cluster is used to perform the experiments. Cluster with more number of nodes may be used in the future to perform the experiments and investigate the impact of Cassandra compaction.

Also, we have used one million operations to conduct the experiment. Experiments with different operation rate needs to be examined to find the impact of Cassandra compaction.

Also, we have used three different workloads, 90%Writes-10%Reads, 50%Writes-50%Reads, and 70%Writes-30%Reads. Experiments with other workloads need to be examined to find the impact of Cassandra compaction.

Also, we have tuned 5 compaction parameter out of the 6 (which can be tuned). Experiments with tuning the parameter MemTable_offheap_space_in_mb needs to be conducted for 90%Writes and 10%Reads workload.

Also, Experiments with tuning the parameters are performed for some selected values. Experiments needs to be carried out for different values of the compaction parameters and also for the combination of those parameters.

And at last, after tuning the parameters, the experiments are conducted using only write heavy workload (90%Writes and 10%Reads). Experiments using the workloads 50%Writes-50%Reads, and 70%Writes-30%Reads are not conducted, which need to be examined in the future.
REFERENCES


