Thesis no:MSEE-2016:38

Performance Comparison of Cassandra in LXC and Bare metal

Container Virtualization case study

Reventh Thiruvallur Vangeepuram

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

Contact Information:
Author(s):
Reventh Thiruvallur Vangeepuram
E-mail: reth15@student.bth.se, revanth.tv@gmail.com

University advisor:
Emiliano Cassalicchio
Professor
DIDD
Blekinge Institute of Technology

Abstract

Big data is a developing term that describes any large amount of structured and unstructured data that has the potential to be mined for information. To store this type of large amounts of data, cloud storage systems are necessary. These cloud storage systems are developed such that they are capable of keeping the data accessible and available to the users over a network. To store big data new platforms are required. Some of the popular big data platforms are Mongo, Cassandra and Hadoop. In this thesis we used Cassandra database system because it is a distributed database and also open source. Cassandra's architecture is master less ring design that is easy to setup and easy to maintain. Apache Cassandra is a highly scalable distributed database designed to handle big data management with linear scalable and seamless multiple data center deployment. It is a NoSQL database system which allow schema free tables so that a data item could have a variable set of columns unlike in relational databases. Cassandra provides with high scalability with no single point of failure.

For the past few years' container based virtualization has been evolving rapidly. Container based virtualization such as LXC have been focused here. Linux Containers (LXC) is an operating system level virtualization method for running multiple isolated Linux systems on a single control host. It does not resemble a virtual machine, but provides a virtual environment that has its own CPU, memory, network, etc. space and the resource control mechanism. In this thesis work performance of Apache Cassandra database has been analyzed between bare metal and Linux Containers(LXC).

A three node Cassandra cluster has been created on both bare metal and Linux container. Assuming one node as seed and Cassandra stress utility tool has been used to test the load of Cassandra cluster. The performance of Cassandra cluster database has been evaluated in bare metal and Linux Container which is the goal of this thesis work.

Linux containers (LXC) are deployed in all the servers. A three node Cassandra database cluster has been created in these servers and also in Linux Container(LXC). Port forwarding is the technique used here for making communication between Cassandra in LXC which is the goal of this thesis work. The performance metrics which determine the performance of Cassandra cluster database are selected according to it. The network configuration parameters are changed according to the behavior of Cassandra. By doing changes in these parameters Cassandra starts running according to the required configuration, after this Cassandra cluster performance will be analyzed. This is done with different write, read and mixed load operations and compared with Cassandra cluster performance on bare metal.

The results of the thesis show an analysis of measurements of performance metrics like CPU utilization, Disk throughput and latency while running on Cassandra cluster in both bare metal and Linux Containers. A quantitative and statistical analysis of performance of Cassandra cluster is compared.

The physical resources utilized by the Cassandra database on native bare metal and Linux Containers (LXC) is similar. According to the results, CPU utilization is more for Cassandra database in Linux Containers. Disk throughput is also more in Linux Containers except in the case of 66% load write operation. Bare metal has less latency compared to Linux Containers in all the scenarios.

Keywords: Cassandra, Container Virtualization, Linux Containers.

ACKNOWLEDGEMENT

I would like to thank my supervisor, Prof. Emiliano Casalicchio. He believed in my efforts and made me learn from my mistakes. In spite of being Senior Professor and editor of online journals, he always found time to help me. He also encouraged be by providing
enough recourses.

Thanks to my mother, Mrs. Radhika, for motivating me with her words. Whenever I felt like giving up, her words kept me moving forward.

Thank you, Prof. Kurt Tutschku, for being great examiner and advisor. His advices and comments on thesis gave motivation to approach scientifically.

ABBREVIATIONS

CPU	Central Processing Unit
CQL	Cassandra Query Language
GB	Giga Bytes
GPS	Global Positioning System
HPC	High Performance Computing
IO	Input-Output
JMX	Java management Extensions
KVM	Kernel based Virtual Machine
LXC	Linux Container
MAC	Media Access Control
NoSQL	Not Only SQL
NIC	Network Interface Card
RAM	Random Access Memory
SSH	Secure Shell
SST	Sorted String Tables
TCP	Transmission Control Protocol
TPS	Transactions per Second

LIST OF FIGURES

Table of Contents

1 INTRODUCTION

Today Big Data systems are the solution for rapidly growing large amounts of data generating from many business organizations and companies. Not only people, mobile devices also generates data constantly while streaming videos, playing games, making purchases, their activity generates data continuously. To store this type of structured or unstructured data there are many database storage systems like NoSQL. Cassandra is one of the most popular big data system among them.

Cassandra is a non-relational and largely distributed database system sometimes referred to as cloud database. Apache Cassandra is a massively scalable open source non-relational database that offers continuous availability, linear performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones (1). Cassandra's architecture is responsible for its ability to scale, perform, and offer continuous uptime. It has a master less "ring" architecture that is easy to setup and easy to maintain (1). In this thesis work we considered a three node Cassandra. In Cassandra, while doing write or read operations it automatically makes stress to all the three nodes equally because all nodes are connected in peer to peer. This makes in Cassandra all nodes play an identical role, there is no concept of a master node with all nodes communicating with each other via a distributed and scalable protocol (1). The wide adoption of Cassandra in Big Data applications is because of its user friendly Cassandra Query Language (CQL), and very efficient write and read access paths that enable critical big data applications to stay always on, scale to millions of transactions per second and handling node and even entire data center failures with ease (2). Cassandra was originally developed and used in Facebook to handle its messenger. Later it became a top level Apache project.

The main goal of the thesis is to evaluate the mixed load, write and read performance of the Cassandra database on bare metal and Linux Containers (LXC). The performance is evaluated for two node Cassandra cluster. By using Cassandra stress utility tool load will be generated and data will be stored in Cassandra memtable. While using this stress utility tool Cassandra database performance will be evaluated. Another part of this thesis work is installing Linux Containers(LXC) in bare metal and repeating the same method as discussed above. Then Cassandra database performance on bare metal and Linux Container(LXC) is compared.

1.1 Thesis Statement

The main aim of the thesis is to evaluate the performance of Cassandra database on Linux Containers and comparing this with Cassandra database on bare metal. A two node Cassandra cluster is created. To evaluate the performance of Cassandra initially load is generated to the cluster and stress utility tool is used to generate three modes of operations like mixed load, write and read. CPU utilization, Disk throughput and latency are evaluated on the servers while running the stress utility tool. For making the most use of the physical resources Cassandra database is deployed on container virtualization which performs less overhead compared to hypervisor technology.

1.2 Background

Cloud computing is evolving day by day. We can store and access data over internet instead of using hard disk. To access this data for long term use we need to store it. This is data is generated from people, devices and network. While streaming videos, playing games, cell phone GPS signals and their activity generates data according to their needs and preferences (3). According to one survey every day we are creating 2.5 quintillion bytes of data and that too 90% of the data in the world today has been created in the last two years. This data is Big Data (4). To store this high voluminous data, we need a scalable and powerful database management system. Due to the increase in demand many companies like Facebook, Apple, Netflix etc. started using the database called Cassandra. Because of its peer to peer architecture allows high performance with linear scalability and no single points of failure (5).

Because of its master less ring architecture it results in an extremely fault tolerant system. Cassandra provides extremely fast, linearly scalable writes. Due to its linear scalability it makes read and write performance very simple (6). Once we have measured our write performance or a single server we can easily calculate how many servers to add to our cluster according to our required performance basis (6). Even under heavy workloads Cassandra delivers higher performance.

Here in this thesis work we study Cassandra performance on bare metal and Linux Containers(LXC) while making write, read and mixed load operations. Linux Containers(LXC) are installed on three nodes each provided. Cassandra is deployed on each container and the selected parameters are changed according to our requirements to run Cassandra. We also propose Cassandra's best performance by comparing in both bare metal and Linux Containers(LXC) at a given load.

1.3 Research Questions

R1) How does Cassandra performs in Linux Containers(LXC) when compared with bare metal?

R2) What is the load used and how many number of nodes in the cluster? What configuration gives the Cassandra's best performance?

R3) Which of the Cassandra parameters effect its performance? What is the CPU Utilization and Disk throughput of Cassandra on physical servers and LXC?

R4) How does Cassandra perform for different load scenarios?

1.4 Scope of implementation

The main idea of this thesis work is to set-up Cassandra cluster and each Cassandra node run in a container and analyzing how its performance varies when compared with bare metal. In this approach, the operating system's kernel runs on the hardware node with many isolated guests are installed on top of it. These isolated guests are called containers (7). Apache Cassandra database system is installed in these containers. Similarly, there are other NoSQL database systems which can be installed in the same environment. Here in this thesis work we created a two node Cassandra cluster but we can increase number of nodes in the cluster. Here we propose the configuration at which Cassandra provides its best performance.

1.5 Objectives

- Understanding Cassandra working procedure
- Deeper analysis and behavior of Cassandra database
- Container Virtualization and networking
- Installing Cassandra database in Linux containers
- Creating Cassandra cluster in Linux containers
- Performance evaluation of Cassandra in different environments

2 RELATED WORK

In this section we discuss some related and significant research works in the field of NoSQL systems and Container virtualization.

2.1 State of the art

2.1.1 Performance of Cassandra

There has been a lot of research in the field of Big Data system in the cloud environment. The research activity ranges between using different big data storage systems, here in this thesis work Cassandra performance on bare metal and Linux Containers(LXC) are compared and best performance determined. Here we also discuss about Cassandra for different write, read and mixed workload operations. The suitable network configuration for Apache Cassandra to run in Linux Containers(LXC) is defined.

In this paper [8], the performance of Cassandra is compared with MySQL and HBase for heavy write operations. Here throughput is selected as performance metric, nGrinder will calculate it in terms of Transactions per Second (TPS). This paper also explains how write operations were performed. It clearly explains how Cassandra scaled up the most amongst the other two databases with fast write speeds.

This paper [9] shows how to design a performance monitoring tool which will help to make decisions to optimize performance of Cassandra database. This is because developers need to have an idea about how database is behaving in different working environments. The performance of Cassandra database in terms of CPU utilization and disk throughput are studied here.

2.1.2 Performance of Containers

This paper [10], compares performance of Cassandra in container based virtualization and hypervisor based virtualization. Here, Linux container(LXC) showed performance in gain when compared to hypervisor based virtualization system Xen. This is because LXC uses "deadline" Linux scheduler, it imposes a deadline on all I/O operations to ensure that no request gets starved. This experiment was conducted in HPC environment. As per the results shown, containers systems showed poor performance isolation for memory, disk and network. According to the analysis, author stated that LXC demonstrates to be the most suitable container based systems for HPC environment.

Containers system level virtualization are becoming a mainstream technology to support cloud and distributed computing applications. In this paper [11], Performance and scalability of kernel modules in Linux container networking was discussed. Here the researcher divided testing part into two scenarios. One is local tests and the other is switched tests. As per the results, for both local testing and switched testing there is no proper difference between ipvlan and macvlan for TCP packets but for UDP macvlan performs well. Here the researcher concluded that macvlan (bridge mode) should be considered for best performance. LXC networking part understood here.

Docker a light weight container level virtualization platform which uses Linux kernel in the background. This paper [12] discusses about performance of Docker containers based on their system performance. The system resource utilization is considered for it. In this paper Docker container architecture was studied and how it is handled to evaluate it performance. The performance of Docker can be compared to the performance of an OS running on bare metal was concluded here due to its good performance.

This paper [13] shows performance evaluation between KVM and LXC two virtualization tools. According to the results obtained in this research LXC was better compared to KVM in most of the scenarios. Because container based virtualization is a good alternative to overcome overhead issues from virtualization.

3 TECHNOLOGICAL OVERVIEW

3.1 Apache Cassandra

Cassandra is an open source distributed database system designed to handle huge amount of structured data and is available under the Apache license. It is designed in such a way that it can handle big data across multiple nodes with no single point of failure. Data is distributed across all nodes in the cluster. Its peer-to-peer distributed system makes easy to address the problem of failures in Cassandra. The communication between all nodes in the cluster will be seen for every second. Data is sequentially written to an in-memory structure when each node captures write activity; this structure is called memtable Once the memory structure is full, next the data will be written to the disk in an SSTable data file. Cassandra periodically consolidates SSTables using the process called compaction [14].

Based on Cassandra architecture client can sent any read and write requests to any node in the cluster. Its key structures and components in Cassandra are explained below.

3.1.1 Node

Node is the place where all our data will be stored. Generally, a node in the cluster is connected with other nodes in the cluster through high internal network. All nodes in the cluster work together even if any node in the cluster fails due to unexpected error as a whole cluster can provide the service. It clearly says all nodes in the cluster are same and there is no any master node. These nodes are peer to peer connected. We can add as many nodes as we want in Cassandra cluster. For example, Apple used 75,000 nodes served Cassandra cluster in 2014.

3.1.2 Data center

A data center is a collection of Racks. It is logical grouping of nodes which separates from another node. The replication strategy called Network Topology is used here to specify number of replicas of the entire keyspace should exist in any given datacenter. A datacenter can be a physical or virtual datacenter. It depends on type of the workloads to use the datacenter. By using separate datacenters, it prevents the Cassandra transactions from being impacted from other workloads to achieve lower latency.

3.1.3 Cluster

Cluster is a collection of datacenters. Cluster allows clients to add a node or delete a node depending on their usage. There is no chance of communication between two clusters.

3.1.4 Commit log

The main purpose of commit log is it has the ability to recreate the memtable after the node crashes or gets rebooted. Because when memtable is full it gets flushed to disk. Writing to commit log is better than just writing to SSTables because SSTables stores rows in sorted order whereas commit log stores updates in the order which they are processed by Cassandra. Cassandra is able to truncate the commit log once all the older data is written to it. When Cassandra start running it has to read the commit log back from that last known point. The write path in Cassandra works in this way drawn below

```
Cassandra Node ----------→ Memtable
                         |           |
                         |           |---→Periodically flush to SSTable
                         |
                         |-------→Commit log
```

3.1.5 SSTable

Data stores in SSTable whenever there is no space in memtable i.e., when the number of keys exceeds the limit or it reaches the time duration. It stores in SSTable, immutable space. This process is called flushing. Once writes are done on SSTable, then we can see the data in data folder. SSTable mainly comprises two files – Index file and Data file. Index file contains bloom filter and key-offset pairs. Cassandra uses bloom filter to save IO when performing the writes. Data file contains actual column data.

3.1.6 keyspaces

Keyspaces in Cassandra is a namespace, it defines data replication on nodes. It has a set of attributes that define keyspace in wide behavior. In Cassandra, the basic attributes we can set for keyspaces are Replication factor. Replication factor refers to the number of replicas of each row of data.

3.2 Cassandra Data Structure

The data in Cassandra database is stored in tables. If any node goes down some part of the data will be unavailable. This problem will overcome by creating copies of data. This copies of data are called replicas. These copies of data are stored on multiple nodes is referred to as replication. Replication of data resembles fault tolerance and reliability.

3.2.1 Cassandra Write Path

Cassandra is a master less ring architecture such that users can connect with any node in a cluster.

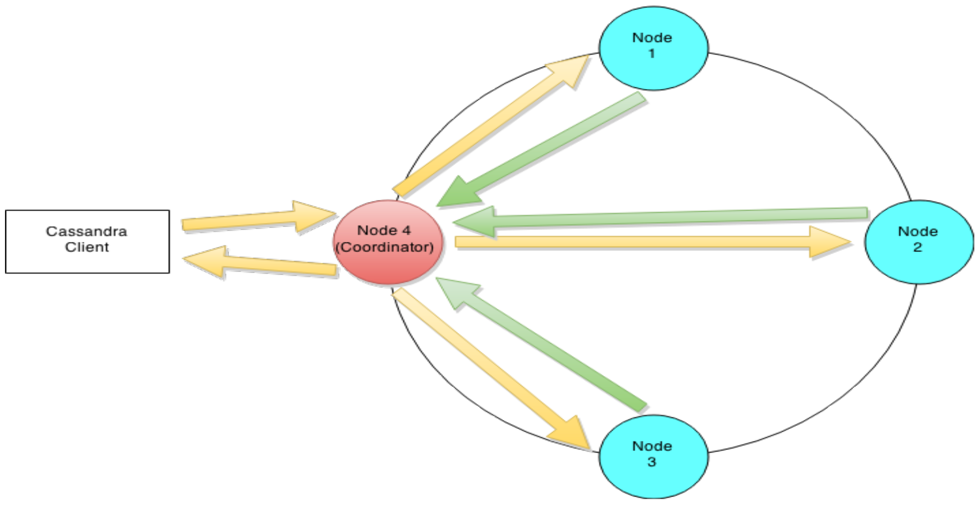


Figure 1 - Data flow in Cassandra cluster [16]

In the above figure Cassandra cluster level interaction for write and read operation is shown. By using either a thrift protocol or CQL clients can interface with a Cassandra node. From the above figure, client has connected to node 4 which acts as a coordinator. Through a messaging service all the inter-node requests are sent in an asynchronous manner. Coordinator forwards the mutation to all the applicable nodes based on the partition key and the replication strategy. Nodes 1, 2 and 3 will act as an applicable node where node 1 is the first replica and nodes 2 and 3 are their subsequent replicas.

Write operation in every node first writes the mutation to commit log and then writes the mutation to memtable. By writing to commit log means it will ensure durability of the write as the memtable is an in-memory structure. It is only written to the disk when the memtable is flushed to disk. The reasons for flushing of memtable to disk will be when it reaches its maximum allocated size in memory, when the number of minutes can stay in memory elapses or it may be when manually flushed by the client.

SSTable (Sorted string table) is an immutable structure. MemTables are flushed into these SSTable. When data from the memtable is lost due to node failure then commit log is used for playback purposes. Because of compaction process SSTables are combined so that related data can be found in a single SSTable. This process makes the operation much faster. In compaction process SSTables are merged together with predefined strategy.

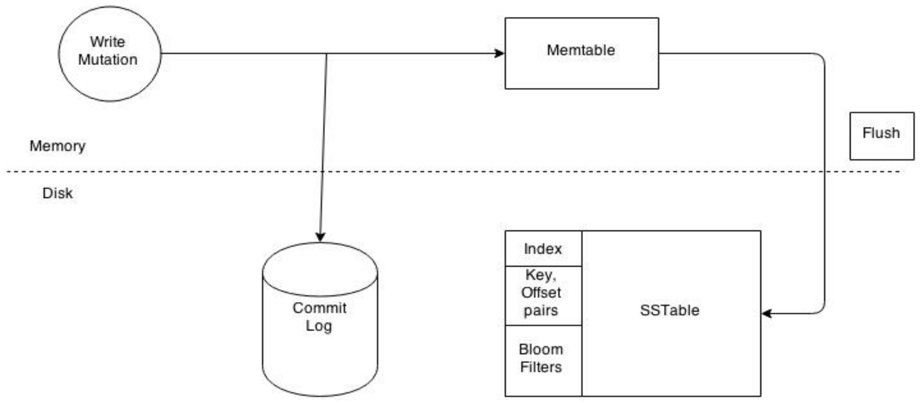


Figure 2 - Cassandra Write Path [16]

3.2.2 Cassandra Read Path

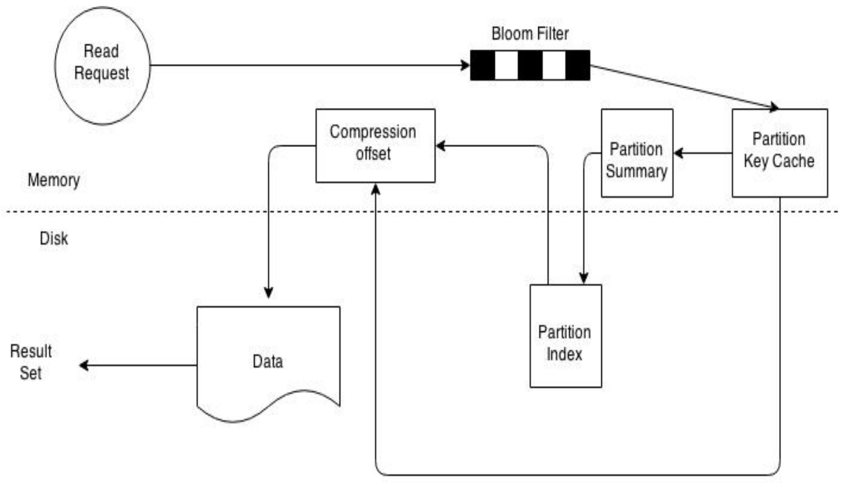


Figure 3 - Cassandra Read Path [16]

A read operation is similar to write operation in Cassandra cluster. By using the write operation client can connect with any node in the cluster. For every read operation a row key must be applied. To determine the first replica coordinator uses the row key.

Node level read operation illustrates about key steps when reading data on a particular node. Every column family stores data in SSTables. Data for each row will be located in SSTables and the memtable. For every read operation Cassandra need to read data from all applicable SSTables. After scanning the memtable for data fragments, this data is then merged and returned to the coordinator.

Read operation becomes more complicated on a per SSTable basis. From the diagram, it illustrates about key steps that take place when reading data from an SSTable. Every SSTable has a bloom filter. This enables to quickly ascertain data for the requested row key. Bloom filter is always held here for the purpose of saving the disk IO. The Coordinator gets all the read requests and decides which nodes to handle read requests. If requests are processed, then it takes data to the client. If this doesn't happen then read request enters key cache memory. Then key cache memory will hold the index of data columns and stored in SSTable. From SSTable the required data will be found in the columns and retrieved from it. The data here is merged and Cassandra looks for time stamp to find data in the disk. This merged data will then be returned to the coordinator.

3.3 Linux Containers (LXC)

Linux Containers (LXC) is light weight virtualization mechanism. It doesn't require any emulation of physical hardware. LXC runs a complete copy of Linux Operating system without the overhead of running a level-2 hypervisor. Linux Container processes and file system are completely visible from the host OS because it shares the kernel with host OS.

3.3.1 Container Networking

There are four major modules currently available and their description is described below in detail.

Veth

This veth kernel module creates a pair of virtual networking devices. They are connected to each other. Veth connection pipes are frequently used in combination with Linux bridges. This provides an easy connection between a namespace and a bridge in default networking namespace. We should remember one thing when running a container with veth network type enabled, it should have one network interface created on the host and the other one will be in the container [15].

OpenVswitch

This kernel module comes as part of the mainline Linux kernel. This is operated by a separate piece of software. OpenVswitch provides a virtual switch, this supports Open Flow. It uses veth pairs this is somewhat similar to Linux bridges [15].

Macvlan

This kernel module enslaves the driver of the Network Interface Card in kernel space. New devices here have their own MAC address and are located within the same broadcast domain as the default driver. Macvlan kernel module has four different modes of operation. They are explained below.

- Private – If their source MAC address matches with one of the Macvlan interfaces then all the incoming packets on the "slave" virtual interface are dropped. This means no Macvlan devices can communicate each other.

- VEPA – While using this we should assume that adjacent bridge returns all frames. This is the place where source and destination local to the macvlan port. Here the bridge is set up as a reflective relay. All the traffic will be forwarded out to the switch even it is destined for us. And again we rely on the switch at the other end to send it back. This mode of process is also called "hairpin mode".

- Bridge - A special bridge called "pseudo bridge" is created here. This bridge forwards traffic using the RAM of the node as buffer. This allows containers to talk each other but isolates pseudo bridged interfaces from the host.

- Passthru – This is implemented in private mode. It passes the packets to the network due to the standard behavior of a switch not to forward packets back to the port they came from [15].

Ipvlan

Ipvlan is similar to macvlan in some ways like enslaving driver of the NIC in kernel space. But in other ways it differs from macvlan like packets sent all get the same MAC address. Based on layer 3 address forwarding to the correct virtual device is done here. Ipvlan module has two modes of operation.

- L2 mode – All the transmit processes is done up to layer 2. This happens in the namespace of the virtual driver. This is because packets are being sent to the default networking namespace for transmit. This causes ARP timeouts. Therefore, device behaves like a layer 2 device.

- L3 mode – Here all the transmit process is done up to layer 3. This also happens in the namespace of virtual driver. The main difference here is packets are being sent to the default network namespace for layer 2 processing and transmit. It doesn't support for broadcast and multicast.

3.3.2 LXC Architecture

Linux Containers to function correctly, it requires several components most of them are provided by the Linux kernel. As seen in the figure Linux kernel comprises Namespaces, cgroups and SELinux. Kernel namespace ensures process isolation and cgroups are employed to control the system resources. SELinux is used to assure separation between host and container and also between the individual containers [16]. Management interface forms a layer between Linux kernel and containers

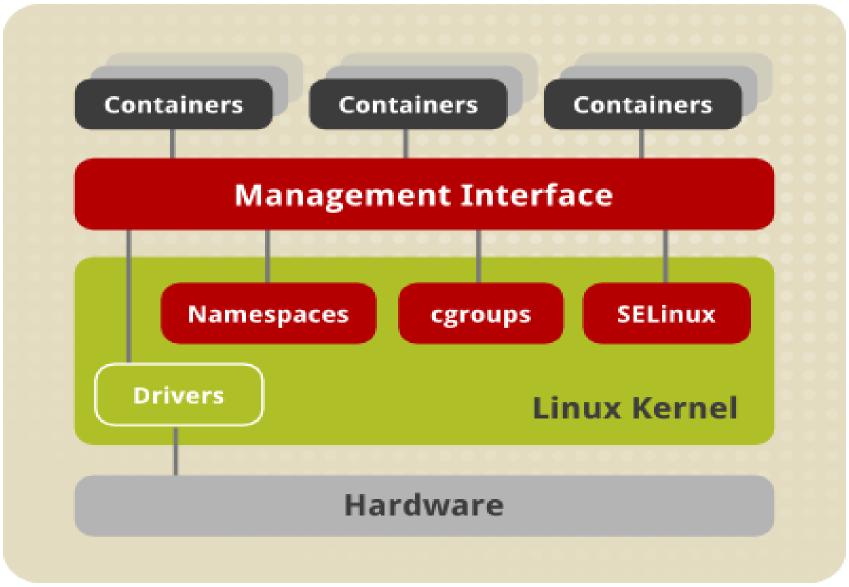


Figure 4 - Linux Container Architecture [17]

Namespaces

By creating separate namespaces for containers kernel provides process isolation. Without creating a problem several containers can use the same resources simultaneously. There are five types of namespaces.

- *Mount namespaces* isolates the set of file system mount points. Processes in different mount namespaces can have different views of the file system hierarchy.
- *UTC namespaces* isolates two system identifiers, they are node name and domain name. This process allows each container to have its own hostname and NIS domain name.
- *IPC namespaces* isolates few inter process communication resources like System V IPC objects and POSIX message queues. This clearly describes that two containers a create shared memory segments with same name. They are not able to interact with other containers.
- *PID namespaces* allows processes to have same PID in different containers. Here container is aware of its own native processes and cannot see the processes running in different parts of the system. Different PID numbers are assigned indeed the host OS is aware of processes running inside the container.
- *Network namespaces* allows container to use separate virtual network stack, loopback device and process space. It also provides isolation of network controllers, system resources associated with networking, firewall and routing tables.

Control groups (cgroups)

The Linux kernel uses cgroups to group processes for the purpose of system resource management. It allocates CPU time, system memory, network bandwidth, or combinations of these among user defined group of tasks [16].

SELinux

SELinux provides secure separation of containers. It integrates with virtual devices by using the sVirt technology.

4 METHODOLOGY

This section explains the method of the experiment in detail. To evaluate and analyze the performance of Cassandra there are different methods of approach to understand. Coming to the measurement and performance evaluation part of the thesis this experimentation has done on a physical model of a system. Physical model of a system is best suited for this analysis over mathematical model, which shows an abstract version with mathematical relations between them. A two node Cassandra cluster is created by changing the required parameters in cassandra.yaml file.

Initially load is generated to the seed node which is considered as a coordinator in Cassandra cluster. Then equal load will be generated to all nodes in the cluster. This is because of its master less ring architecture. After that Cassandra stress utility tool is executed from the load generator with the given mode of operation i.e. write, read and mixed load one after the other. While executing the stress utility tool from the load generator node, sar and iostat commands are executed at the same time in the seed node to measure the CPU utilization and Disk throughput of the Cassandra database in the cluster. This methodology is implemented to measure the performance metrics in both bare metal and in Linux Containers (LXC). In Linux Containers, before creating Cassandra cluster port forwarding must be done from the host servers. This process is discussed further in section 4.4.

4.1 Experimentation

In this thesis work, the experiment was done in two ways. One is evaluating performance of Cassandra database on physical server (bare metal) and the other is performance of Cassandra database on Linux Container (LXC). Both the experiments are done in the same servers therefore they use shared physical resources. The below work flow gives a simple idea of Cassandra load generation and Cassandra stress utility tool functions where Cassandra in 10th node and Cassandra in 12th node are in cluster.

```
Cassandra in 6th Node ----------→ Cassandra in 10th node (194.47.131.211)
   (194.47.131.207)                       (Coordinator)
                  |
                  |
                  |-------------------→Cassandra in 12th node (194.47.131.213)
```

4.1.1 Cassandra stress tool

The Cassandra stress tool is a java based tool used for stress testing utility for basic benchmarking and load testing in a Cassandra cluster. Keyspaces are created by using this stress tool. This tool creates a keyspace called keyspace1 with in that table called standard1. It depends on what type of table being tested. These are created automatically when we run the stress test for the first time. These can be reused on subsequent runs unless we drop these keyspace1 by using CQL. Here in each server CPU utilization and disk throughput are evaluated on their respective servers by using sar tool and iostat tool.

4.1.2 SAR tool

System Activity Report (SAR) is a Unix command used for system V-derived system monitor command used to report on various system loads which includes CPU activity, device load, memory, network. The systat package provides sar tool including with iostat which are system performance utilities. The sar command writes to standard output the contents of selected cumulative activity counters in the operating system [18]. This tool collects, report or save system activity information. Here it used to collect the CPU usage on servers and Linux container. This tool takes a snap shot of the system at regular periodic intervals. The performance characteristics such as CPU utilization, memory usage, interrupt rate, etc. are gathered by using this tool. This tool gives the performance metrics like CPU utilization at the application level, CPU utilization while executing at the user level with nice priority, percentage of CPU utilization in idle form. The below command is used to evaluate CPU utilization which generates average value for 30 seconds i.e. 40 values in 1200 seconds.

```
$ sar -u 30 41 | awk '{print $8 "\t" $9}' > filename.txt
```

4.1.3 Iostat tool

Iostat tool is a command line tool used to report CPU statistics and input/output statistics for devices and partitions. This command monitors system input/output device loading by observing the activity of the devices in relation to their average transfer rates. The reports generated by the iostat command can be used to monitor the system configuration to better balance the input/output load between physical disks [19].
Iostat command provides statistics concerning the time since the system was booted. This tool generates CPU utilization report and Device utilization report. By using this tool, we are evaluating the disk utilization of the server and Linux container. While running this iostat command the following sections will be seen [19]. The below command is used to generate Disk throughput for the duration of 20 minutes.

```
$ iostat -d 30 41 | grep sda | awk '{print $4}' > filename1.txt
```

- tps – This indicates the number of transfers per second that were issued to the device. Here a transfer is an I/O request to the device
- kB_read/s – This indicates the amount of data read from the device
- kB_wrtn/s – This indicates the amount of data written to the device
- kB_read – This indicates the total number of kilobytes read
- kB_wrtn – This indicates the total number of kilobytes written

4.2 Setup

Make sure that few prerequisites have been installed. They are Java python and java native access because Cassandra is a java based database system. Java Open JDK 7 is must for running Cassandra without any problem.

4.2.1 Cassandra package

In this thesis Cassandra 3.0.8 version is used because of its stable version. Before installing this we should update the servers and Linux container. This package is installed through command line interface.

4.2.2 Cassandra cluster

Cassandra is package is installed in all the servers and Linux containers we are using. By changing the required parameters in Cassandra.yaml file it allows to start running Cassandra database. We are using a two node cluster and one node as seed. We should change the local host address to the required IP addresses in Cassandra.yaml file. This is done on all the three nodes. Changes in Cassandra.yaml file are done after stopping the Cassandra. We should not do all these changes while Cassandra database is running. The figure below shows a simple topology of Cassandra database running in a native bare metal server.

The server configuration details are as follows

Table 1 - Server Configuration Details

Operating system	Ubuntu 14.04 LTS (GNU/Linux 3.19.0-49-generic x86_64)
RAM	23 GB
Hard -disk	279.4 GB
Processor	12 cores, 2 threads per core → 24 theoretical cores
Cassandra version	3.0.8
Cassandra-stress tool	2.1

4.3 Performance evaluation of Cassandra in bare-metal

In this thesis, we have used three hosts (one source and two destination hosts). Cassandra 3.0.8 is installed in them. Cassandra stress tool which is a command line tool comes with Cassandra package. This stress tool generates load on cluster, cqlsh utility. A python based command line client for executing CQL commands for managing a cluster.

To evaluate the performance analysis of Cassandra database write, read and mixed load operations are considered. One important thing here is all the servers should have the same configuration of software, hardware and network used. After each iteration each server should have same RAM and hard disk to ensure for high integrity of results.

After running each iteration Cassandra stress tool creates a keyspace called keyspace1 and within the tables standard1 or counter1 in each of the nodes. These are created automatically when we run this for first time and are reused subsequently until and unless we drop the keyspace using CQL. Before testing the load of the database a write operation is done to insert data into it. To evaluate the load on each server sar and iostat are used.

Cluster creation is already discussed in section 4.2.2 by changing the seed address, listen address, rpc address and broadcast address in Cassandra.yaml file to the ip address of the host node. One of the node ip address is set to the seed address to form Cassandra cluster. This allows the nodes to communicate with each other and form the cluster. The below command generates load on Cassandra cluster for the given ip address. And next command uses stress utility tool for the duration of 20 minutes for the mode of operation given. To evaluate 66% load 150 threads are given.

```
$ ./cassandra-stress write n=50000000 -node 194.47.131.211
```

```
$ ./cassandra-stress mixed ratio\(write=1,read=3\) duration=20m cl=ONE -pop dist=UNIFORM\(1..50000000\) -rate threads\=450 -node 194.47.131.211;
```

```
$ ./cassandra-stress mixed ratio\(write=1,read=3\) duration=20m cl=ONE -pop dist=UNIFORM\(1..50000000\) -rate threads\=150 -node 194.47.131.211;
```

To measure Cassandra database performance on load generated node sar tool and iostat tool are used. Sar takes snapshots at regular intervals. %idle is considered here because it shows percentage of the time CPU was idle. By subtracting this with 100 it gives percentage of CPU's usage. Iostat reports input/output for devices and partitions. This tool monitors input/output device loading by observing the active devices in relation to their transfer rates. Disk resources utilized are collected by using iostat command. kB_wrtn/s value of the iostat tool gives us the amount of data written per second to the disk.

By using stress tool data is written to the cluster. This makes to push data to the nodes. Here three cases are considered on this data set. Mixed operation, read operation and write operation for a duration of 20 minutes, while doing these operations on the other side of the nodes CPU utilization and Disk throughput are recorded by using sar and iostat commands. For the duration of 20 minutes' average values of CPU utilization and Disk throughput are for and interval of 30 seconds is taken for the servers in the cluster. Latency value and total time taken by the write and read request from the stress server are noted. The figure below shows a simple topology of Cassandra database running in a native bare metal server.

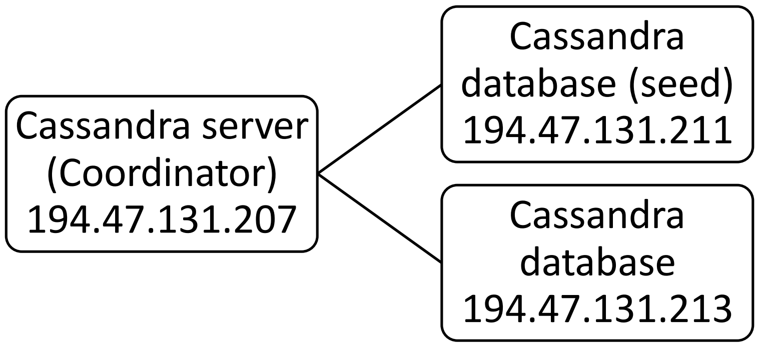

Cassandra server (Coordinator) 194.47.131.207
Cassandra database (seed) 194.47.131.211
Cassandra database 194.47.131.213

Figure 5 - Work flow for Cassandra database in bare metal

4.4 Performance evaluation of Cassandra in Linux Container (LXC)

4.4.1 Configuration

Linux container is installed through terminal command. It installs with default setting and configuration. The following commands below shows how to start and run Linux container. By default, LXC creates a private network namespace for each container.

```
$ sudo apt-get install lxc
```

```
$ sudo lxc-create –n genie –t ubuntu
```

The above command shows how to create Linux Container named “genie”. ‘–t’ refers to template. Ubuntu template is used to create container.

```
$ sudo lxc-start –n genie –d
```

To run the container in the background detached from the console the above command is used. –d is the parameter here used to run the container detached from the console.

```
$ sudo lxc-attach –n genie
```

This command attaches to the container named genie and we can enter into that for further working in it. Here Cassandra 3.0.8 package is installed through command line given below. It is extracted to use it.

```
$ sudo wget http://www-us.apache.org/dist/cassandra/3.0.8/apache-cassandra-3.0.8-bin.tar.gz
```

For running Cassandra in the container we should change the Cassandra.yaml file configuration according to the Linux container Ip addresses in all the two nodes. This is the similar process as we discussed above in section 4.3

Port forwarding is the process used here. This makes communication between the containers and the bare metal servers. This process enables Cassandra in containers to listen to the destination port which is assigned in the given command. To make port forwarding iptables rules should be installed in Linux container and it should be empty.

```
$ sudo apt-get install iptables
```

```
$ iptables -t nat -L -n -v
```

```
$ iptables -t nat -A PREROUTING -p tcp -i br0 -d 194.47.131.211 --dport 7000 -j
DNAT --to 10.0.3.116:7000
```

```
$ sudo iptables -A FORWARD -p tcp -d 10.0.3.116 --dport 7000 -j ACCEPT
```

```
$ iptables -t nat -A PREROUTING -p tcp -i br0 -d 194.47.131.211 --dport 9042 -j
DNAT --to 10.0.3.116:9042
```

```
$ sudo iptables -A FORWARD -p tcp -d 10.0.3.116 --dport 9042 -j ACCEPT
```

From the above commands 10.0.3.116 is the ip address of the container genie. Port '7000' is the internode communication port in Cassandra and '9042' is the CQL native transport port in Cassandra. This enables the Linux container to run Cassandra database. The same process is applied to another container in other node. And for making Cassandra cluster seed address Is set to the address of one node. Listen address, rpc address are set to ip address of Linux container and broadcast address is set to ip address of the respective node in which container is present. The diagram below shows a simple topology of Cassandra database running in a Linux Container.

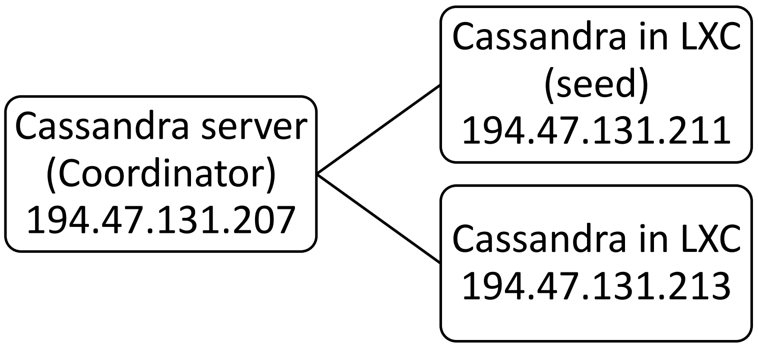

Cassandra server (Coordinator) 194.47.131.207
Cassandra in LXC (seed) 194.47.131.211
Cassandra in LXC 194.47.131.213

Figure 6 - Work flow for Cassandra in Linux Container

4.4.2 Performance evaluation

We considered three modes of operation. They are mixed load, write and read operations. These operations are done once all the nodes form the Cassandra cluster. All the hardware resources of the servers give access to the Cassandra database in the container as there is no other application running for the resources.

While running any one of the operations in write, read and mixed load sar and iostat commands are executed on the other terminal to record CPU utilization and disk throughput of the Cassandra database in the Linux container. The command sar shows the % idle value which gives us the percentage of CPU utilization by subtracting it from 100. The iostat command gives disk throughput in kB_wrtn/s. The procedure is same as what we did in performance evaluation in bare metal which is discussed in section 4.3. The figure below shows a simple topology of Cassandra database running in a native bare metal server. The below commands are used to generate 11 GB data on the cassandra cluster and running cassandra stress utility tool for three modes of operation. To evaluate 66% load 150 threads are given.

```
$ ./cassandra-stress write n=50000000 -node 194.47.131.211

$ ./cassandra-stress mixed ratio\(write=1,read=3\) duration=20m cl=ONE -pop dist=UNIFORM\(1..50000000\) -rate threads\=450 -node 194.47.131.211;

$ ./cassandra-stress mixed ratio\(write=1,read=3\) duration=20m cl=ONE -pop dist=UNIFORM\(1..50000000\) -rate threads\=150 -node 194.47.131.211
```

5 RESULTS

In this section Cassandra database performance on different scenarios are shown and explained in terms of CPU utilization and Disk throughput.

5.1 CPU utilization

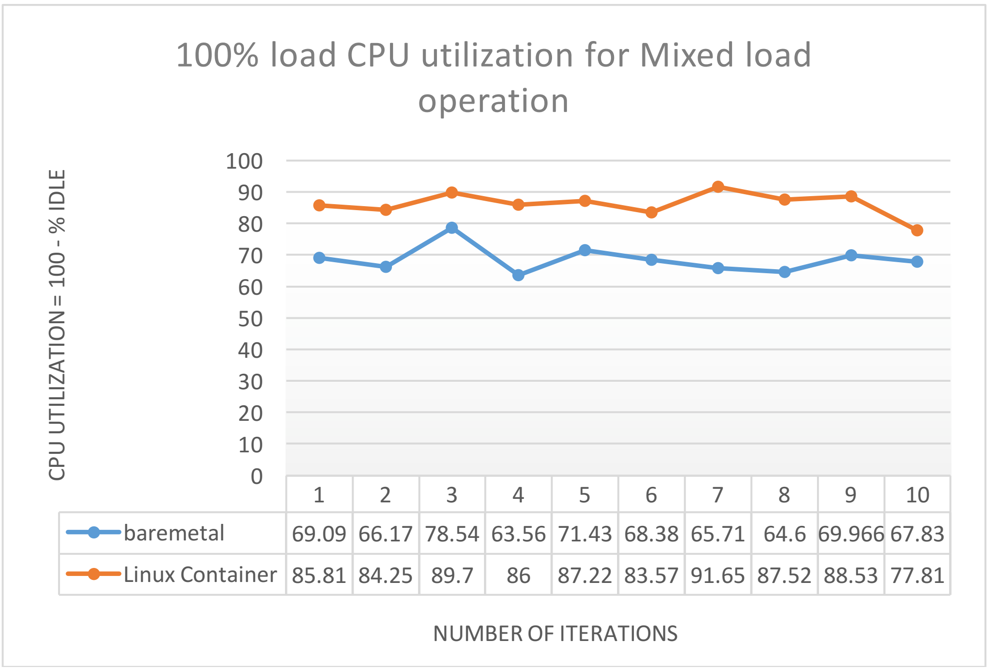
CPU utilization of Cassandra database on the servers is evaluated by running sar command tool for three different scenarios i.e. for mixed load, write and read operations. These are done by sung Cassandra stress tool. These operations are done for a duration of 20 minutes. Sar command tool is executed in such a way that it gives average value of %idle of CPU usage for every 30 seconds. That means we can collect 40 average values of % idle for the total duration of 1200 seconds. This value shows how much time the CPU spends on user processes and system processes.

5.1.1 Mixed load operation

This operation is a mix of 1 write and 3 read processes in Cassandra stress tool for the duration of 20 minutes.

For 100% load:

The below figure shows the graph for 100% load CPU Utilization for Mixed load operation. 100% load is determined by op rate while running mixed load operation which stresses on 11GB data in the Cassandra cluster and 450 threads are given in the Cassandra stress command. The below graph shows the average values of CPU utilization for 10 iterations. CPU utilization of Cassandra on bare metal and Cassandra on Linux Containers is compared here. It shows Cassandra utilizes more CPU usage on Linux Containers than Cassandra on bare metal.



	1	2	3	4	5	6	7	8	9	10
baremetal	69.09	66.17	78.54	63.56	71.43	68.38	65.71	64.6	69.966	67.83
Linux Container	85.81	84.25	89.7	86	87.22	83.57	91.65	87.52	88.53	77.81

Figure 7 - 100% load CPU utilization for Mixed load

The highest CPU utilization on Linux Containers is 91.65%. Highest CPU utilization on bare metal is 78.54%. There are some reasons for this difference. Sometimes nodes become unresponsive if for several seconds this causes the clusters to start thrashing the load around. The reason might be Cassandra uses more CPU cycles.

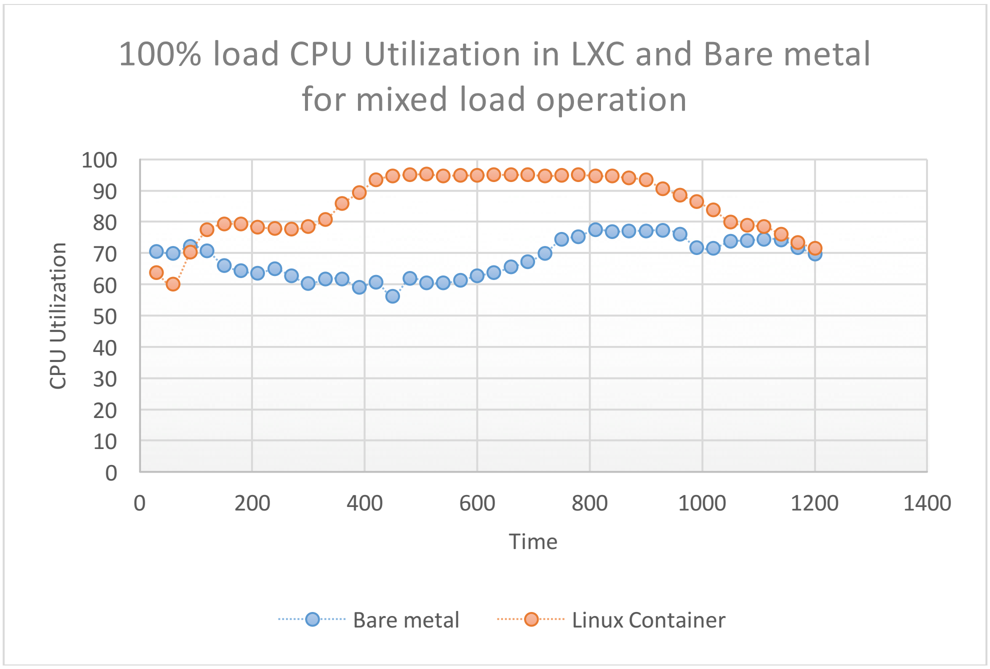
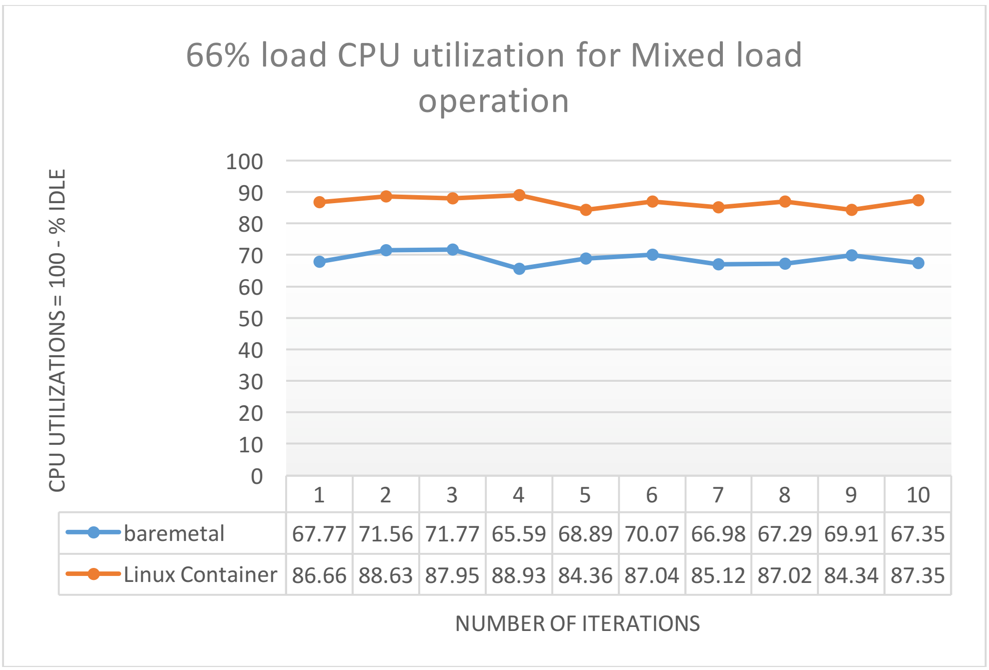


Figure 8 - 100% load CPU utilization in LXC and bare metal for mixed load operation

Sar tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Sar tool is executed in such a way to give value every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed such that CPU utilization is more in case of Linux Containers.

For 66% Load:



	1	2	3	4	5	6	7	8	9	10
baremetal	67.77	71.56	71.77	65.59	68.89	70.07	66.98	67.29	69.91	67.35
Linux Container	86.66	88.63	87.95	88.93	84.36	87.04	85.12	87.02	84.34	87.35

Figure 9 - 66% load CPU utilization for Mixed load operation

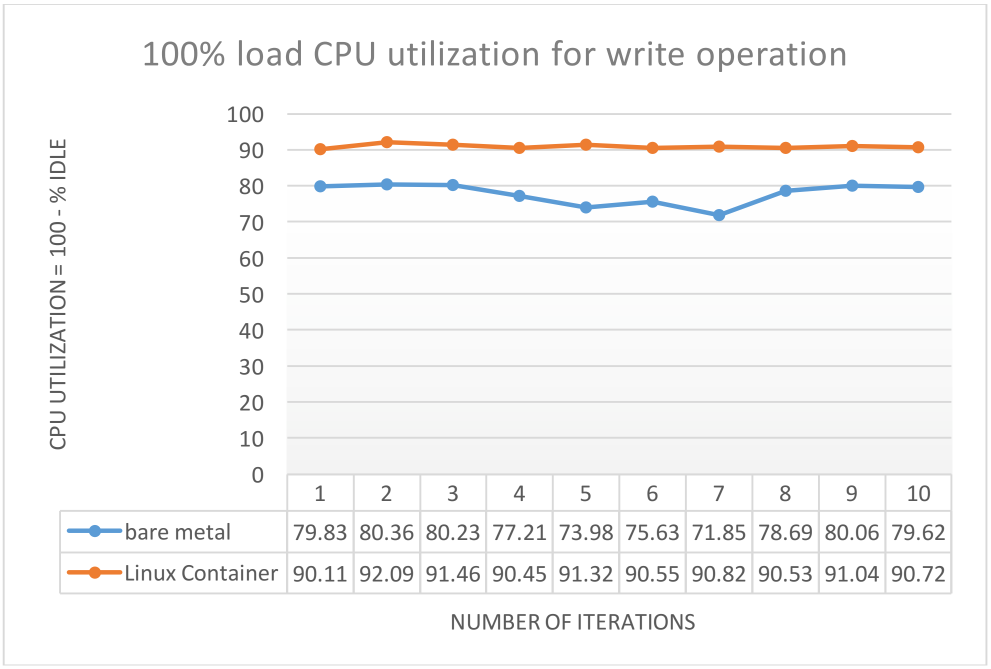
The above figure shows the graph for 66% load CPU utilization for Mixed load operation. 66% load is determined by op rate while running mixed load operation which stresses on 11GB data in the Cassandra cluster and 150 threads are given in the Cassandra stress command. The above graph shows the average values of CPU utilization for 10 iterations. It shows Cassandra utilizes more CPU usage on Linux Containers than Cassandra on bare metal in both 100% load and 66% load cases. Highest CPU usage on Linux Container is 88.93%. Highest CPU usage on bare metal is 71.77%. In the case of Linux Containers there might be a heavy data volume request traffic per Cassandra node. Because of read and write operations it naturally creates very high load on the cluster. This means columns being read; columns being compacted will quickly become old. This old generation will fill up faster causes high CPU utilization.

5.1.2 Write operation

This operation is done by giving write process in Cassandra stress tool for 20 minutes.

For 100% load and 66% load:

The below figure shows the graph for 100% load CPU utilization for write operation. 100% load is determined by the oprate while running the write operation using Cassandra stress tool. The graph shows the average values of CPU utilization for 10 iterations. It shows Cassandra utilizes more CPU usage in Linux Container than bare metal.



	1	2	3	4	5	6	7	8	9	10
bare metal	79.83	80.36	80.23	77.21	73.98	75.63	71.85	78.69	80.06	79.62
Linux Container	90.11	92.09	91.46	90.45	91.32	90.55	90.82	90.53	91.04	90.72

Figure 10 - 100% load CPU utilization for write operation

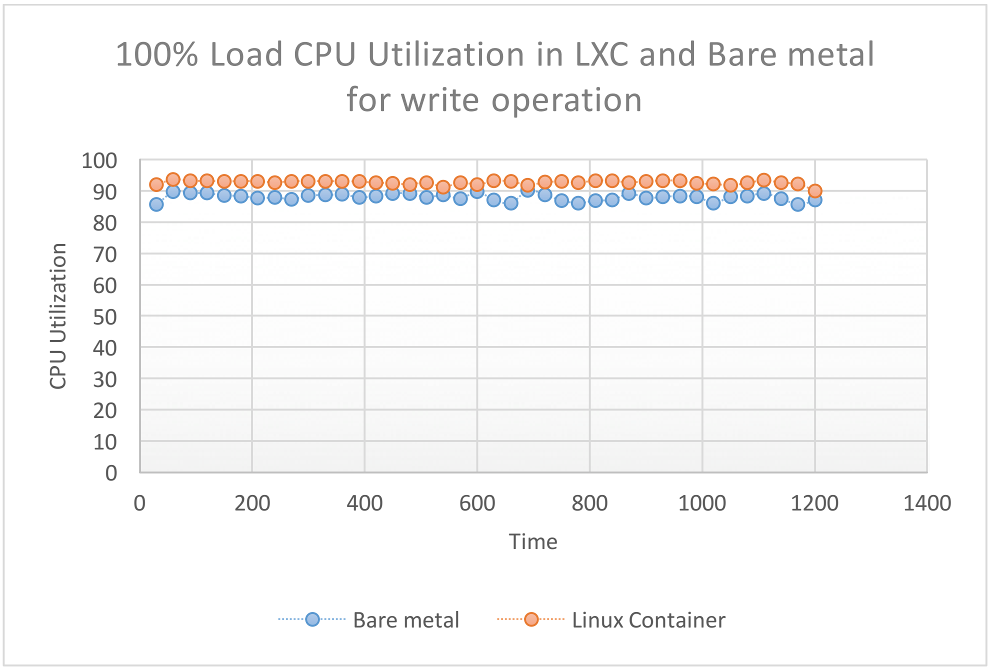
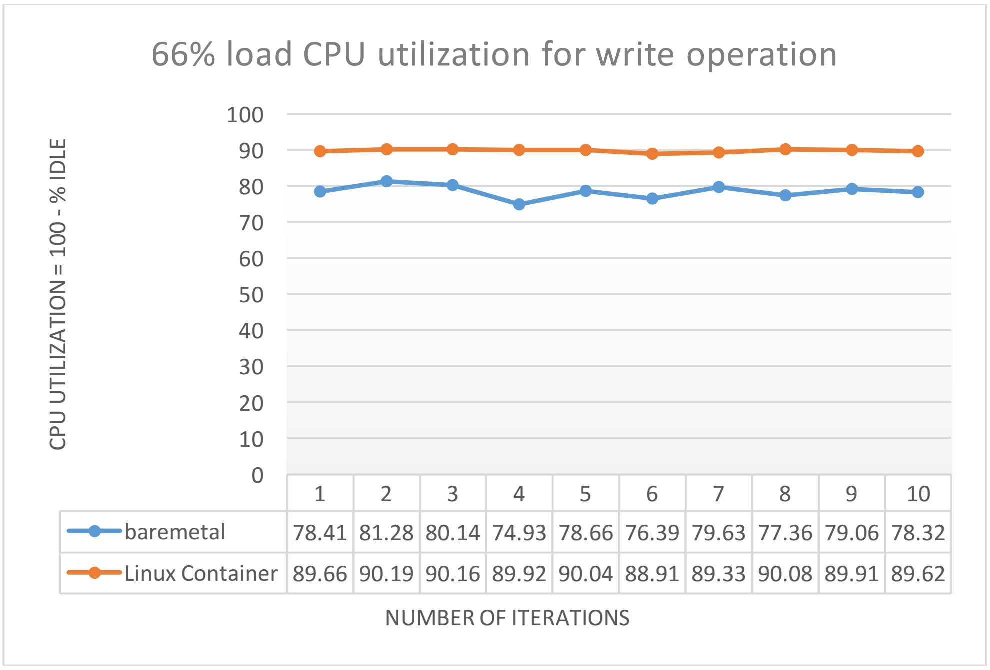


Figure 11 - 100% load CPU utilization in LXC and bare metal for write operation

Sar tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Sar tool is executed in such a way to give value every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed such that CPU utilization is more in case of Linux Containers.



	1	2	3	4	5	6	7	8	9	10
baremetal	78.41	81.28	80.14	74.93	78.66	76.39	79.63	77.36	79.06	78.32
Linux Container	89.66	90.19	90.16	89.92	90.04	88.91	89.33	90.08	89.91	89.62

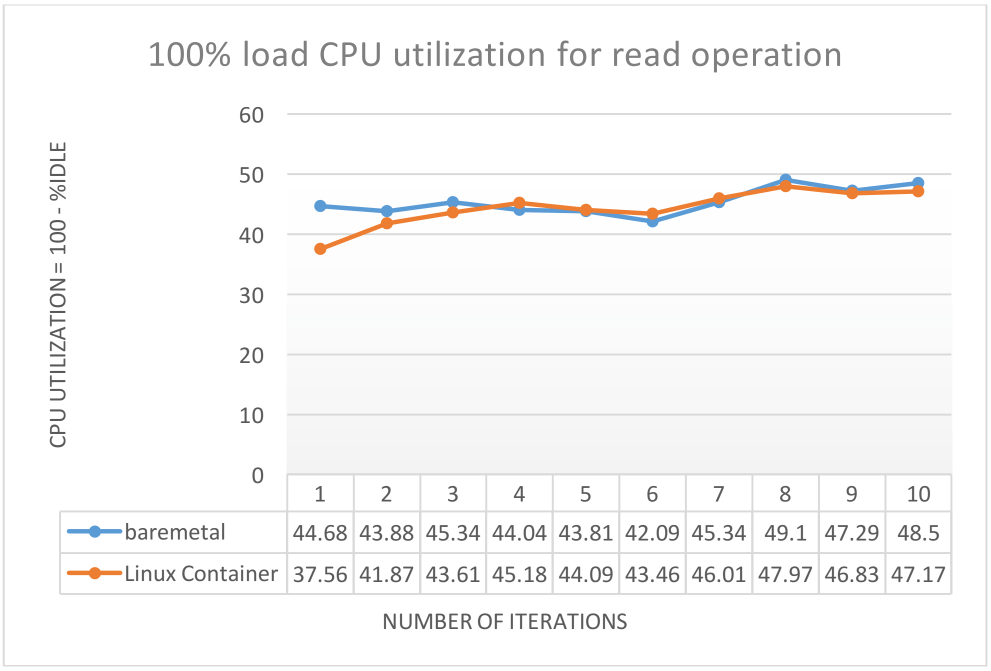
Figure 12 - 66% load CPU utilization for write operation

From the above figure, for 100% load the highest CPU usage on Linux Container is 92.09%. Highest CPU usage on bare metal is 80.36%. For 66% load the highest CPU usage on Linux Container is 90.19%. Highest CPU usage on bare metal is 81.28%. CPU utilization in both the cases it is almost same. When Cassandra uses heavy data volume and request traffic per node gradually CPU usage increases. This causes very high CPU cycles. This might be the reason for very high CPU utilization.

5.1.3 Read Operation

This operation is done by giving Read process in Cassandra stress tool for 20 minutes.

For 100% load and 66% load:



	1	2	3	4	5	6	7	8	9	10
baremetal	44.68	43.88	45.34	44.04	43.81	42.09	45.34	49.1	47.29	48.5
Linux Container	37.56	41.87	43.61	45.18	44.09	43.46	46.01	47.97	46.83	47.17

Figure 13 - 100% load CPU utilization for read operation

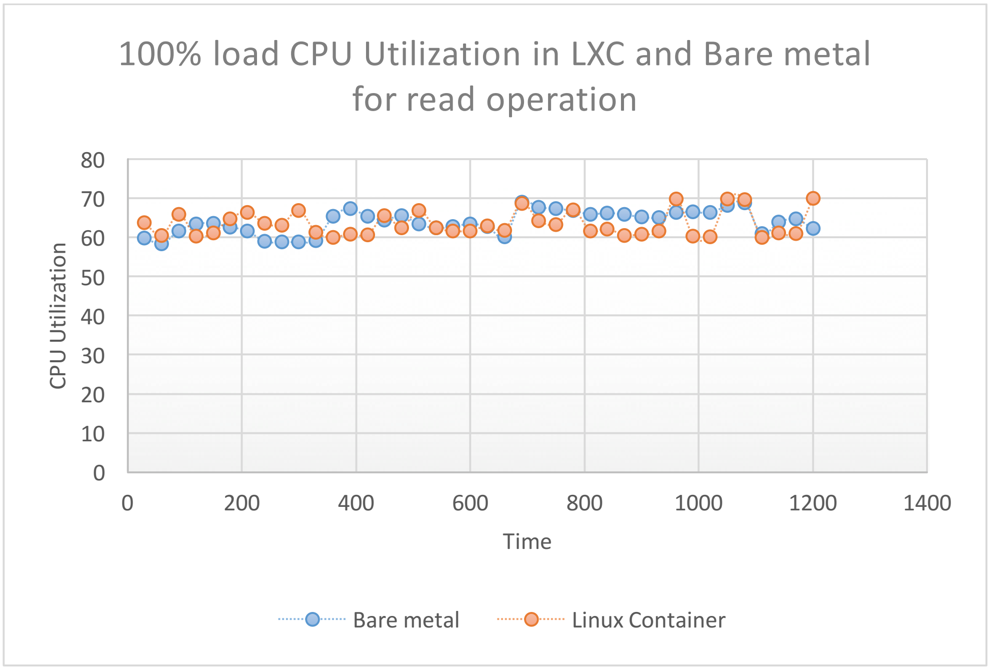
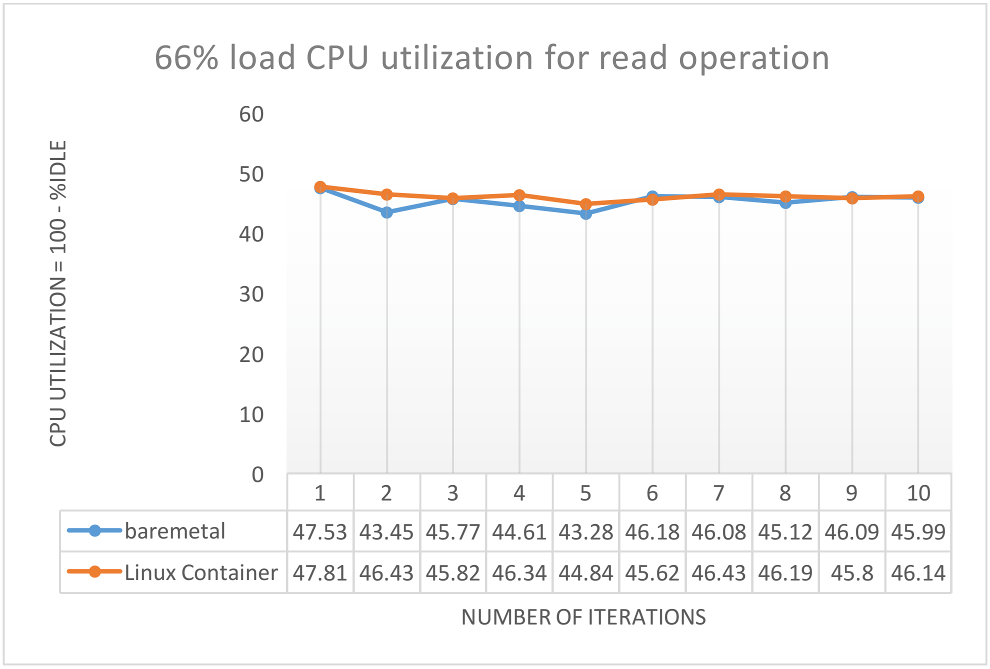


Figure 14 - 100% load CPU utilization in LXC and bare metal for read operation

Sar tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Sar tool is executed in such a way to give results every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed such that CPU utilization is more in case of Linux Containers.



	1	2	3	4	5	6	7	8	9	10
baremetal	47.53	43.45	45.77	44.61	43.28	46.18	46.08	45.12	46.09	45.99
Linux Container	47.81	46.43	45.82	46.34	44.84	45.62	46.43	46.19	45.8	46.14

Figure 15 - 66% load CPU utilization for read operation

From the above figure, there is no difference between CPU utilization in bare metal and Linux Container in both the cases 100% load and 66% load read operation. But when we compare these values with mixed load and write operations the CPU utilized here in read operation is very less. There is very less request traffic on Cassandra cluster then it uses less CPU cycles compared to write and mixed load operation.

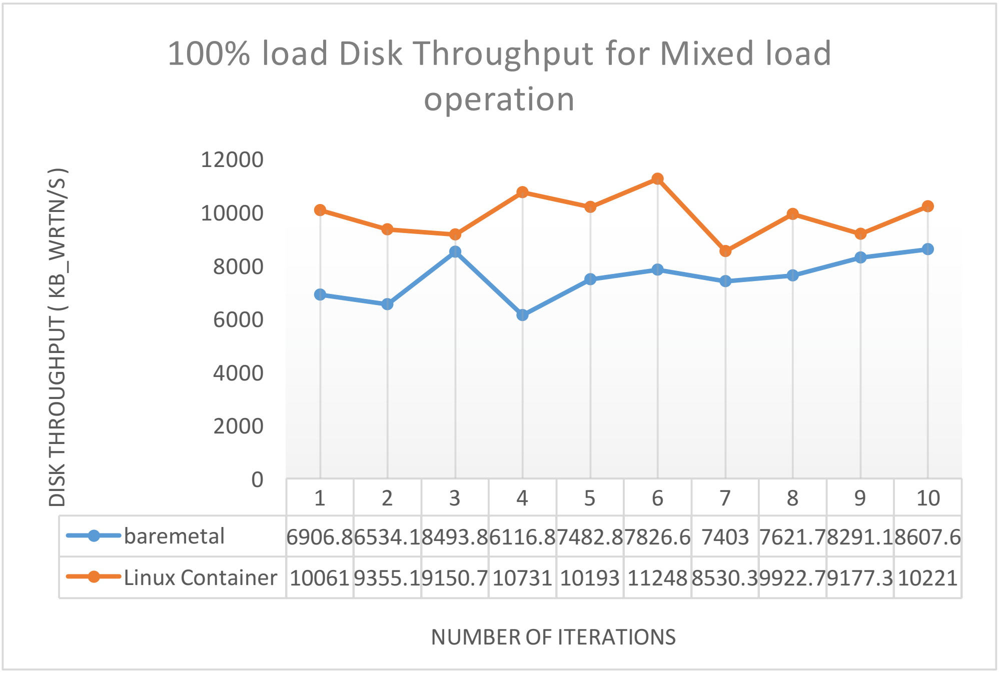
5.2 Disk Throughput

Disk Throughput of Cassandra database is evaluated by running iostat command tool for three different scenarios i.e. for mixed load, write and read operations. These operations are done for a duration of 20 minutes. Iostat command is executed in such a way that disk usage is listed for every 30 seconds. That means we will get 40 values for 1200 seconds. Disk usage is shown in kB_wrtn/s. For each iteration disk throughput value is taken as average and plotted.

5.2.1 Mixed load operation

This operation is a mix of 1 write and 3 read processes in Cassandra stress tool for the duration of 20 minutes.

For 100% load and 66% load:



	1	2	3	4	5	6	7	8	9	10
baremetal	6906.8	6534.1	8493.8	6116.8	7482.8	7826.6	7403	7621.7	8291.1	8607.6
Linux Container	10061	9355.1	9150.7	10731	10193	11248	8530.3	9922.7	9177.3	10221

Figure 16 - 100% load Disk Throughput for mixed load operation

From the above figure, Disk throughput is higher in Linux containers. Highest value is 11247.69kB_wrtn/s. The highest value for Cassandra in bare metal is 8607.56kB_wrtn/s. 100% load and 66% load is determined from oprate value while running Cassandra stress utility tool. In Cassandra, there is a compaction strategy which merges multiple memtables after SSTables being flushed. This causes more disk usage.

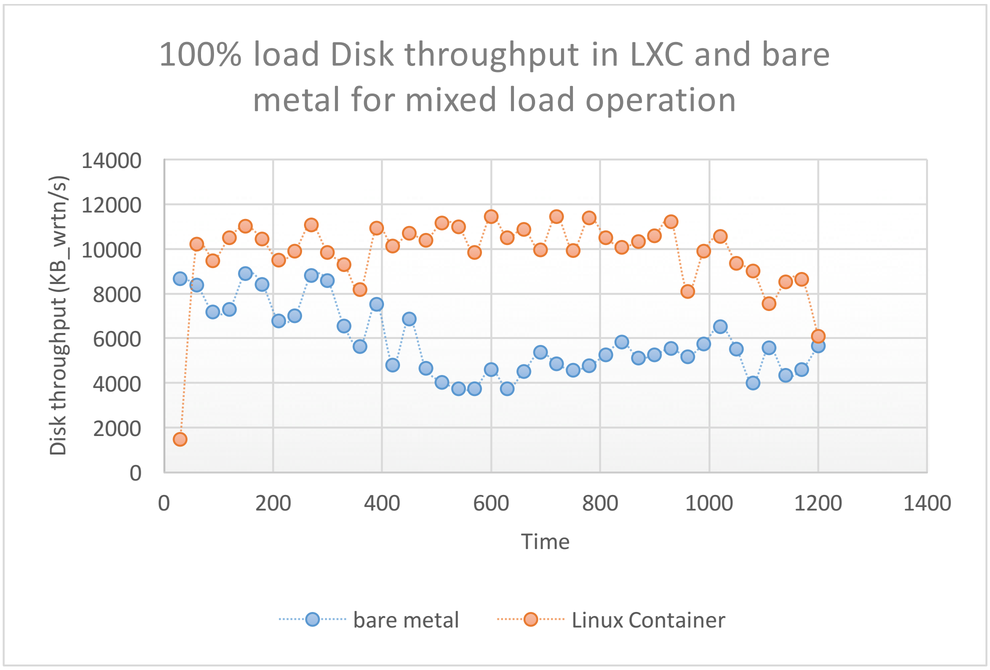
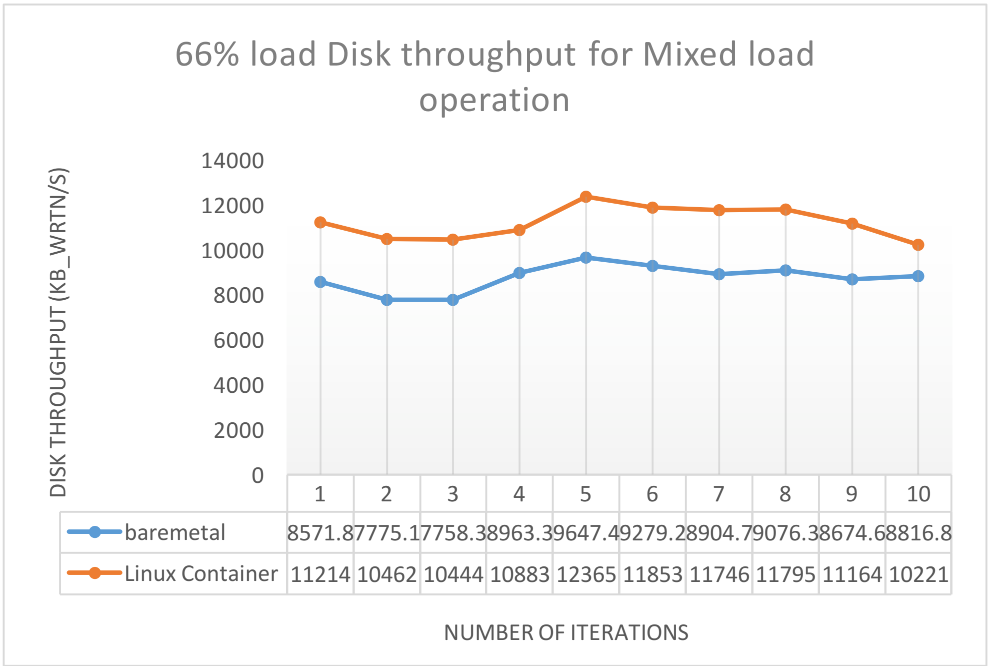


Figure 17 - 100% load Disk throughput in LXC and bare metal for mixed load operation

Iostat tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Iostat tool is executed in such a way to give value every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed such that Disk throughput is more in case of Linux Containers.



	1	2	3	4	5	6	7	8	9	10
baremetal	8571.8	7775.1	7758.3	8963.3	9647.4	9279.2	8904.7	9076.3	8674.6	8816.8
Linux Container	11214	10462	10444	10883	12365	11853	11746	11795	11164	10221

Figure 18 - 66% load Disk throughput for mixed load operation

From the above figure, Disk throughput is higher in Linux Containers. Highest value in Linux Containers is 12364.63kB_wrtn/s. The highest value for Cassandra in bare metal is 9647.38kB_wrtn/s.

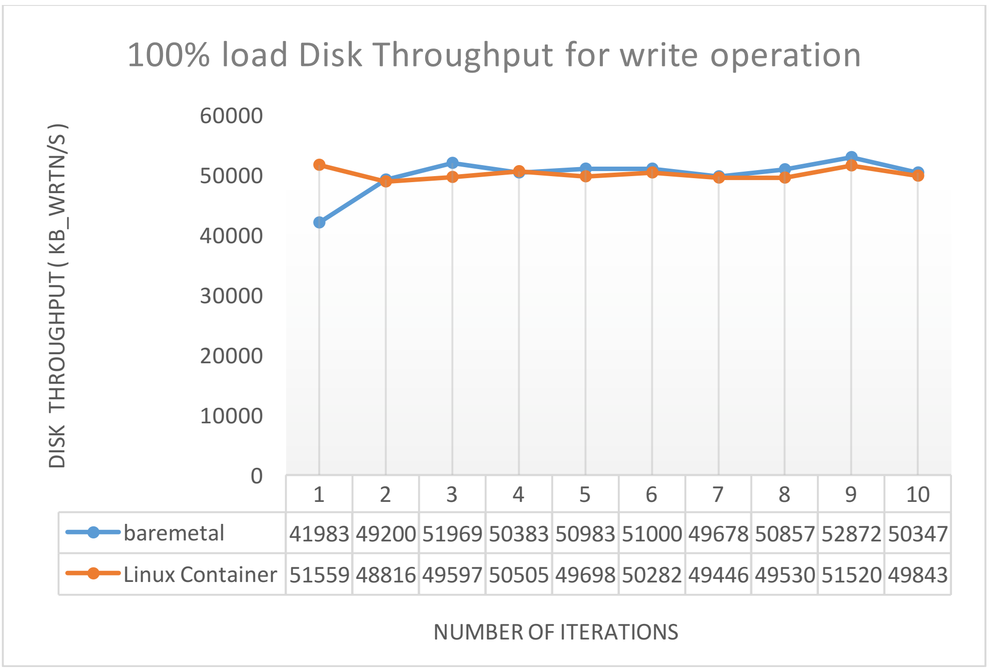
Cassandra on Linux Containers has higher disk throughput values because disk usage values may be higher than expected because of heavy writing and reading processes including with building SSTables as the final product of compaction processes.

5.2.2 Write Operation

This operation is done by giving write process in Cassandra stress tool for 20 minutes.

For 100% load and 66% load:

The figure below shows the graph for 100% load Disk throughput for write operation. 100% load and 66% load is determined from the oprate value while running the cassandra stress utility tool. The graph below shows the average values of Disk throughput for 10 iterations. For 100% load operation, disk throughput is almost same in both bare metal and Linux Containers. Coming to 66% load, Cassandra in bare metal has highest disk utilization than in Cassandra in Linux Containers. The reason for this might be heavy write processes to the disk for Cassandra in bare metal. It also has effect from compaction strategy which merges memtables after SSTables being flushed. This process utilizes more disk usage. In normal operations maintaining a free disk space of 30% is recommended.



	1	2	3	4	5	6	7	8	9	10
baremetal	41983	49200	51969	50383	50983	51000	49678	50857	52872	50347
Linux Container	51559	48816	49597	50505	49698	50282	49446	49530	51520	49843

Figure 19 - 100% load Disk throughput for write operation

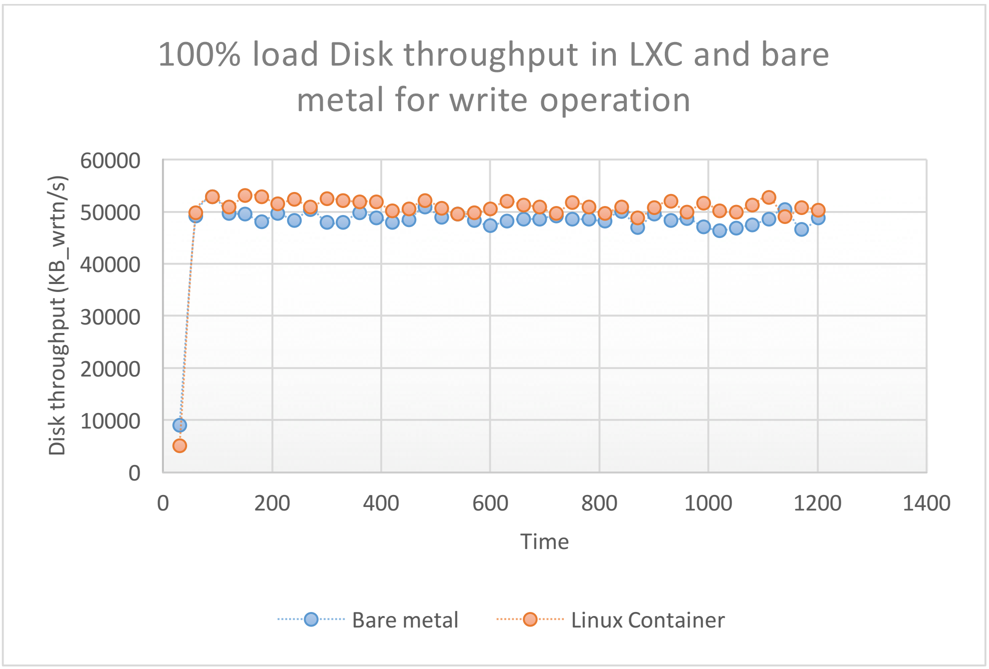
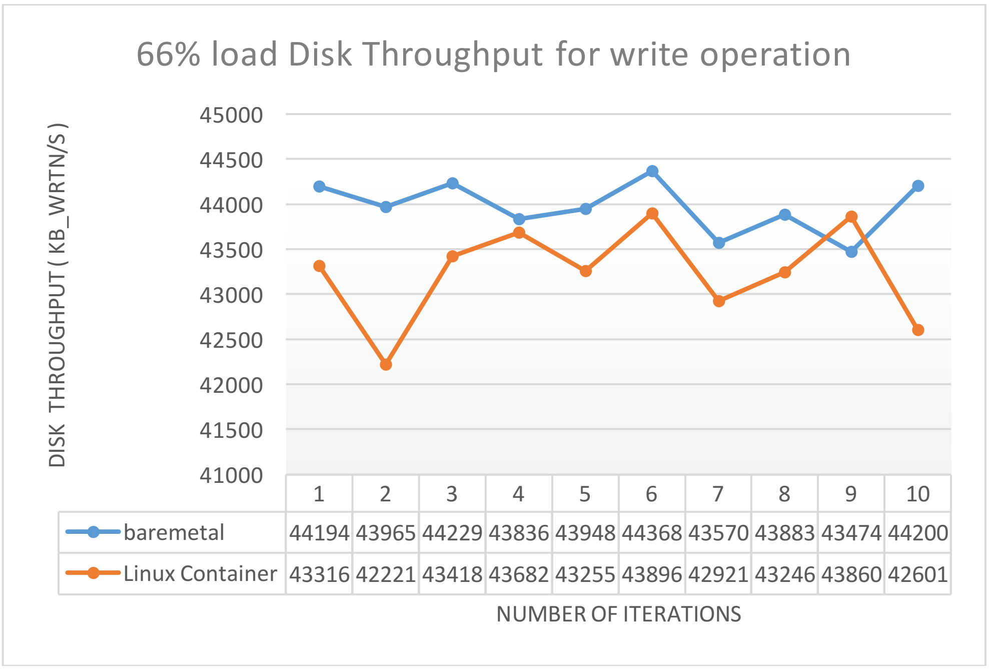


Figure 20 - 100% load Disk throughput in LXC and bare metal for write operation

Iostat tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Iostat tool is executed in such a way to give value every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed.



	1	2	3	4	5	6	7	8	9	10
baremetal	44194	43965	44229	43836	43948	44368	43570	43883	43474	44200
Linux Container	43316	42221	43418	43682	43255	43896	42921	43246	43860	42601

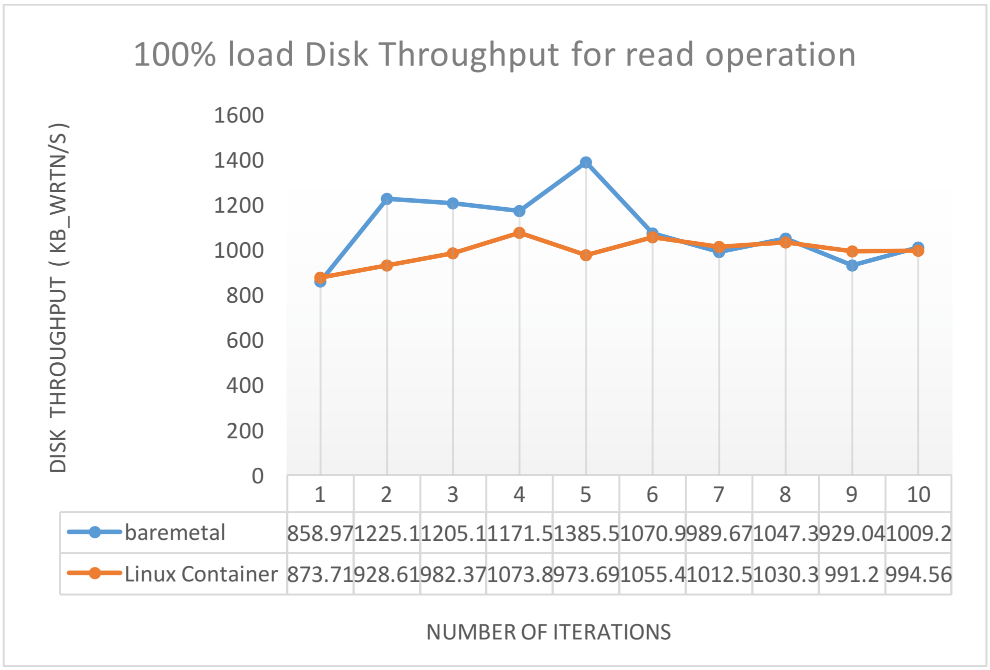
Figure 21 - 66% load Disk throughput for write operation

For 66% load operation, the highest disk throughput in bare metal is 44896.15kB_wrtn/s. Highest disk throughput in Linux Containers is 43860.2kB_wrtn/s. Here Disk throughput is more for bare metal.

5.2.3 Read operation

This operation is done by giving Read process in Cassandra stress tool for 20 minutes. When compared to mixed load and write operations disk throughput in read operation is very less. This is because of very less read processes to the disk.

For 100% load and 66% load:



	1	2	3	4	5	6	7	8	9	10
baremetal	858.97	1225.1	1205.1	1171.5	1385.5	1070.9	989.67	1047.3	929.04	1009.2
Linux Container	873.71	928.61	982.37	1073.8	973.69	1055.4	1012.5	1030.3	991.2	994.56

Figure 22 - 100% load Disk throughput for read operation

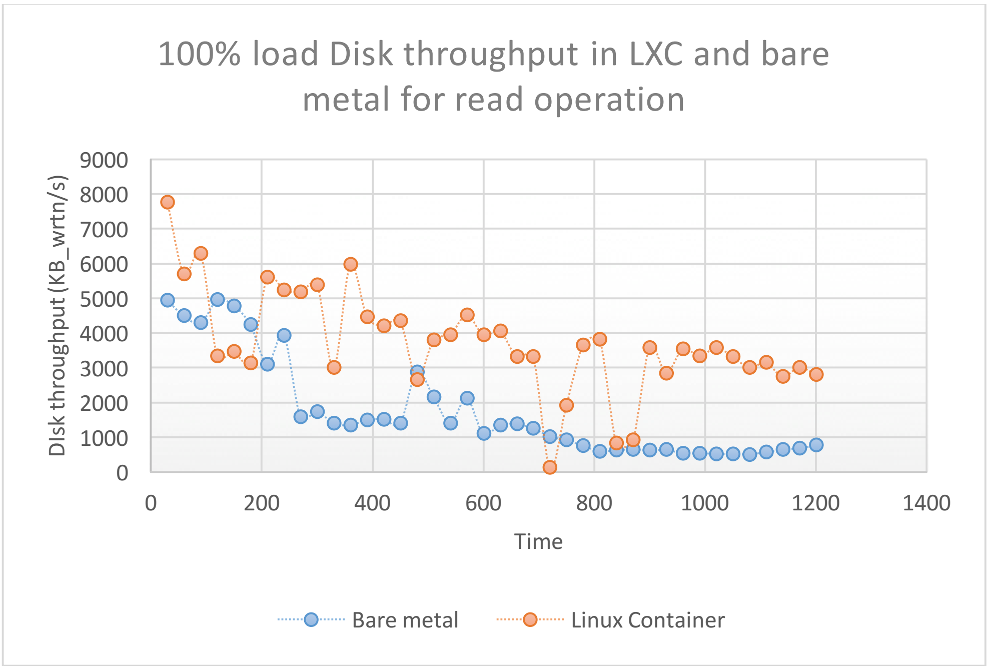
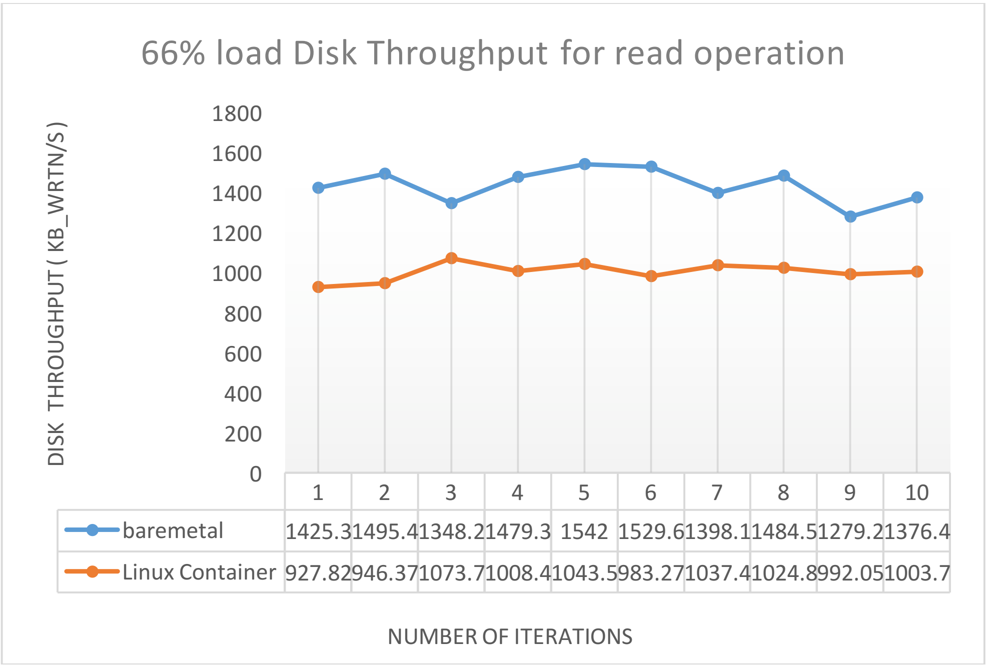


Figure 23 - 100% load Disk throughput in LXC and bare metal for read operation

Iostat tool is executed to run for 20 minutes' duration. In the above figure each value represents the average of 10 iterations from 30 to 1200 seconds. Iostat tool is executed in such a way to give value every time after 30 seconds, therefore 40 average values are plotted. From the above two graphs the same trend has been observed such that Disk throughput is more in case of Linux Containers and it was similar for few iterations.



	1	2	3	4	5	6	7	8	9	10
baremetal	1425.3	1495.4	1348.2	1479.3	1542	1529.6	1398.1	1484.5	1279.2	1376.4
Linux Container	927.82	946.37	1073.7	1008.4	1043.5	983.27	1037.4	1024.8	992.05	1003.7

Figure 24 - 66% load Disk throughput for read operation

There is no compaction strategy in read operation. The reason for this is after the memtable is flushed SSTables are not allowed to write it again. If a row is not in memtable, a read of the row needs to lookup in all multiple SSTable files. This causes read operation in Cassandra a little bit slower than write operation.

5.3 Latency

Here latency can be defined as time taken by the process to generate the load and also generate a response from the Cassandra cluster. Latency values are noted after running the stress utility tool in load generator (6th node). As the load increases the latency for all operations increases because we will find more number of responses for requests per second.

5.3.1 Mixed Load Operation

For 100% load:

This process is a mix of 1 write and 3 read operations executed by using Cassandra stress utility tool in the load generator. Latency is noted after running the stress utility tool in load generator. For 100% load 450 threads are used.

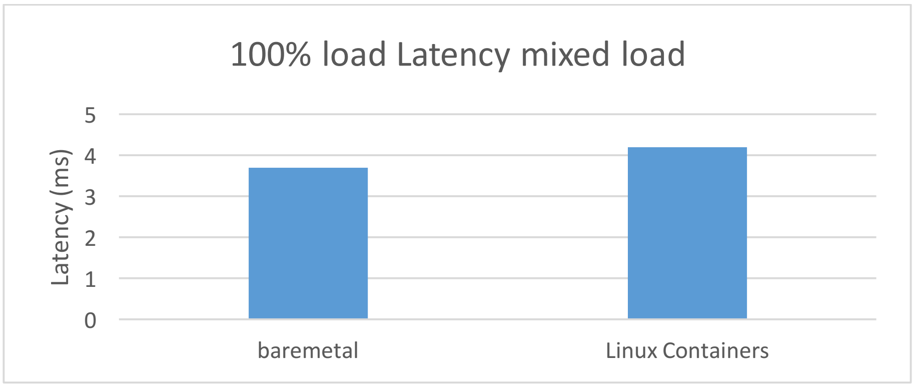


Figure 25 - 100% load latency for mixed load operation

For 66% load:

This process is a mix of 1 write and 3 read operations executed by using Cassandra stress utility tool in the load generator. Latency is noted after running the stress utility tool in load generator. For 66% load 150 threads are given in the stress tool utility command.

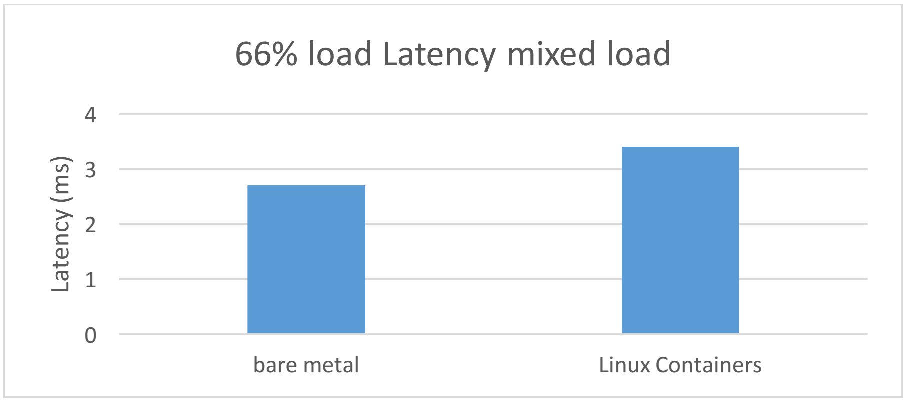


Figure 26 - 66% load latency for mixed load operation

5.3.2 Write Operation

For 100% load

This process is done by write operation executed by Cassandra stress utility tool in the load generator for 20 minutes. Latency is noted after running the stress utility tool in load generator. For 100% load 450 threads are used.

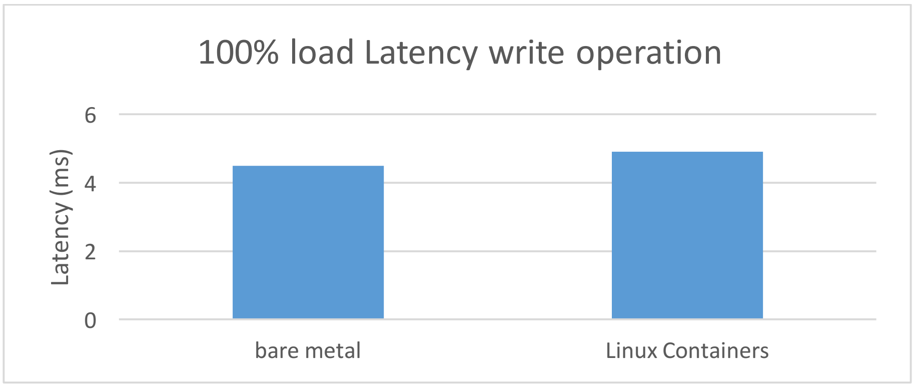


Figure 27 - 100% load latency for write operation

For 66% Load

This process is done by write operation executed by Cassandra stress utility tool in the load generator for 20 minutes. Latency is noted after running the stress utility tool in load generator. For 66% load 150 threads are used.

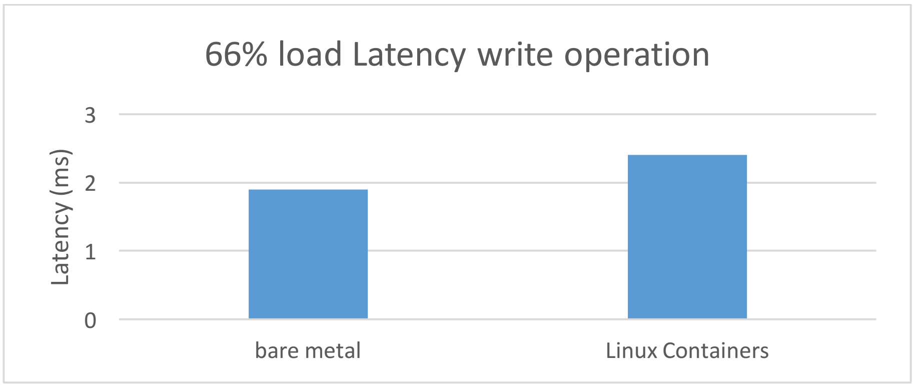


Figure 28 - 66% load latency for write operation

5.3.3 Read Operation

For 100% load

This process is done by read operation executed by Cassandra stress utility tool in the load generator for 20 minutes. Latency is noted after running the stress utility tool in load generator. For 100% load 450 threads are used. Sometimes there will be more latency because of its slower process.

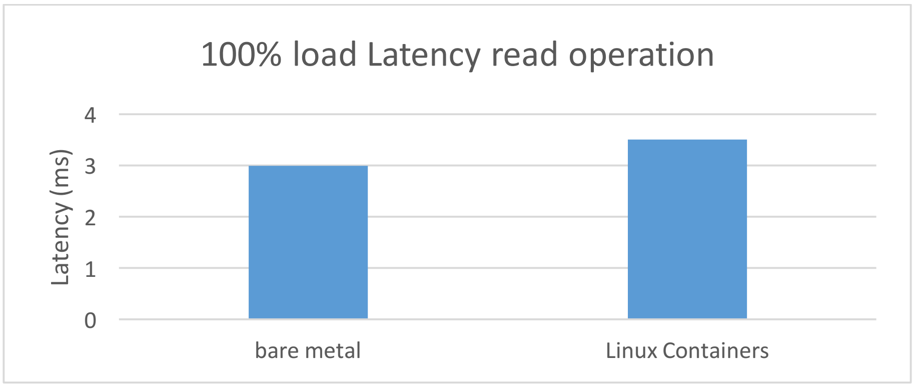


Figure 29 - 100% load latency for read operation

For 66% load

This process is done by read operation executed by Cassandra stress utility tool in the load generator for 20 minutes. Latency is noted after running the stress utility tool in load generator. For 66% load 150 threads are used. Sometimes there will be more latency because of its slower process

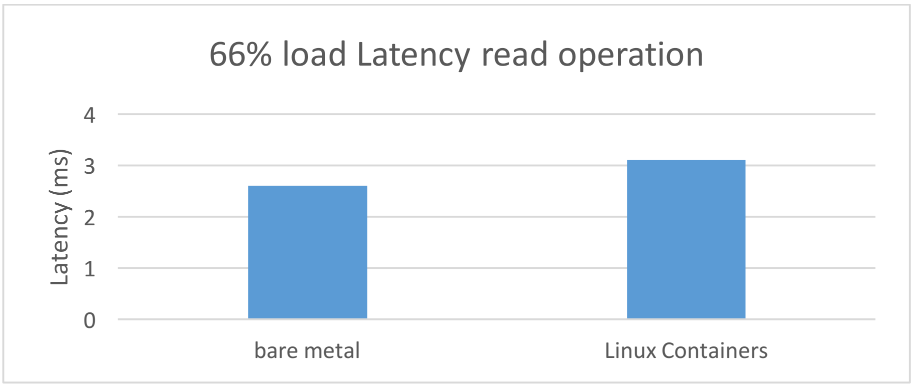


Figure 30 - 66% load latency for read operation

6 ANALYSIS AND DISCUSSION

For both the cases 100% load and 66% load in mixed load and write operations we observe very high CPU Utilization for Cassandra database in both bare metal and Linux Container. The reason for this high CPU utilization might be Cassandra uses up so much CPU cycles. This can also be shown as data volume and request traffic per Cassandra node increases. Then CPU Utilization increases.

While running mixed load operation which have a mix of read and write operations for millions each per node, default heap settings will not work here. This naturally creates very high load on the Cassandra cluster. This means that columns being read, column being compacted, key caches, memtables etc. will quickly become old generation. This old generation will fill up faster and will potentially have high CPU Utilization.

CPU Utilization and Disk Throughput in read operation is almost similar for both Linux Container and bare metal and less compared to mixed load and write operations. The reason for this is after the memtable is flushed SSTables are not allowed to write it again. Therefore, if a row is not in memtable, a read of the row needs to lookup in all multiple SSTable files. This is the reason why read operation in Cassandra is slower than write operation. So Disk throughput for read operation is also very less.

Coming to Disk throughput in write and mixed load operations there is compaction strategy which merges multiple memtables this process uses more disk usage. There is no compaction strategy in read operation which causes less disk usage. In normal operations maintaining a free disk space of 30% - 50% is recommended. This is for having sufficient available space for Cassandra to perform compaction strategy which uses additional disk space.

7 CONCLUSION AND FUTURE WORK

The main aim of this thesis is to evaluate mixed load, write and read performance of Cassandra database and comparing its CPU utilization and Disk Throughput on bare metal and Linux Containers (LXC). According to the results, CPU utilization is more for Cassandra database in Linux Containers. We observed overhead in case of Linux Containers. Disk throughput is also more in Linux Containers except in the case of 66% load write operation. This means bare metal performs less CPU utilization, less Disk throughput in except one scenario. From these results we observe Cassandra database performs better on bare metal because it utilizes less CPU usage. Coming to latency, bare metal has less latency compared to Linux Containers in all scenarios. This is also a main reason to say Cassandra on bare metal performs better.

By comparing the results, the advantage of Cassandra on physical servers (bare metal) is its CPU utilization is almost 20 percent less in both mixed load, write operation and it is same in read operation. The only advantage for Linux Containers observed in this thesis work is Disk throughput for cassandra is more when compared to physical server (bare metal) in mixed load operation. For write and read operation disk throughput is more for physical servers (bare metal). For both 100% load and 66% load the same trends has been observed.

As future work, we can evaluate performance technologies of Cassandra database in cloud systems and High Performance Computing (HPC). We can also evaluate performance of compaction strategies in Cassandra database in Linux Containers. Container virtualization is improving day by day we can implement the same method of this experiment on different platform.

REFERENCES

[1] "A Brief Introduction to Apache Cassandra | DataStax Academy: Free Cassandra Tutorials and Training." [Online]. Available: https://academy.datastax.com/resources/brief-introduction-apache-cassandra. [Accessed: 14-Sep-2016].

[2] A. Chebotko, A. Kashlev, and S. Lu, "A Big Data Modeling Methodology for Apache Cassandra," in *2015 IEEE International Congress on Big Data (BigData Congress)*, 2015, pp. 238–245.

[3] "Big data analytics – actionable insights for the communication service provider - wp-big-data.pdf." [Online]. Available: http://www.ericsson.com/res/docs/whitepapers/wp-big-data.pdf. [Accessed: 14-Sep-2016].

[4] "IBM - What is big data?," 07-Sep-2016. [Online]. Available: https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html. [Accessed: 14-Sep-2016].

[5] "Why should I use Cassandra?," *DataStax*. [Online]. Available: http://www.datastax.com/2012/01/why-should-i-use-cassandra. [Accessed: 14-Sep-2016].

[6] "5 reasons why you should use Cassandra - Exponential.io." [Online]. Available: http://exponential.io/blog/2015/01/13/5-reasons-why-you-should-use-cassandra/. [Accessed: 14-Sep-2016].

[7] "What is container-based virtualization (operating system-level virtualization)? - Definition from WhatIs.com," *SearchServerVirtualization*. [Online]. Available: http://searchservervirtualization.techtarget.com/definition/container-based-virtualization-operating-system-level-virtualization. [Accessed: 14-Sep-2016].

[8] V. D. Jogi and A. Sinha, "Performance evaluation of MySQL, Cassandra and HBase for heavy write operation," in *2016 3rd International Conference on Recent Advances in Information Technology (RAIT)*, 2016, pp. 586–590.

[9] P. Bagade, A. Chandra, and A. B. Dhende, "Designing performance monitoring tool for NoSQL Cassandra distributed database," in *2012 International Conference on Education and e-Learning Innovations (ICEELI)*, 2012, pp. 1–5.

[10] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. D. Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," in *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013, pp. 233–240.

[11] J. Claassen, R. Koning, and P. Grosso, "Linux containers networking: Performance and scalability of kernel modules," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 713–717.

[12] P. E. N, F. J. P. Mulerickal, B. Paul, and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," in *2015 International Conference on Control Communication Computing India (ICCC)*, 2015, pp. 697–700.

[13] D. Beserra, E. D. Moreno, P. T. Endo, J. Barreto, D. Sadok, and S. Fernandes, "Performance Analysis of LXC for HPC Environments," in *2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, 2015, pp. 358–363.

[14] "Architecture in brief." [Online]. Available: https://docs.datastax.com/en/cassandra/2.0/cassandra/architecture/architectureIntro_c.html. [Accessed: 14-Sep-2016].

[15] J. Claassen, R. Koning, and P. Grosso, "Linux containers networking: Performance and scalability of kernel modules," in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 2016, pp. 713–717.

[17] "Chapter 1. Introduction to Linux Containers - Red Hat Customer Portal." [Online]. Available: https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/7/paged/getting-started-with-containers/chapter-1-introduction-to-linux-containers. [Accessed: 14-Sep-2016].

[16] “Introduction to Apache Cassandra’s Architecture - DZone Database,” *dzone.com*. [Online]. Available: https://dzone.com/articles/introduction-apache-cassandras. [Accessed: 14-Sep-2016].

[18] “sar(1) - Linux man page.” [Online]. Available: http://linux.die.net/man/1/sar. [Accessed: 14-Sep-2016].

[19] “iostat(1) - Linux man page.” [Online]. Available: http://linux.die.net/man/1/iostat. [Accessed: 14-Sep-2016].

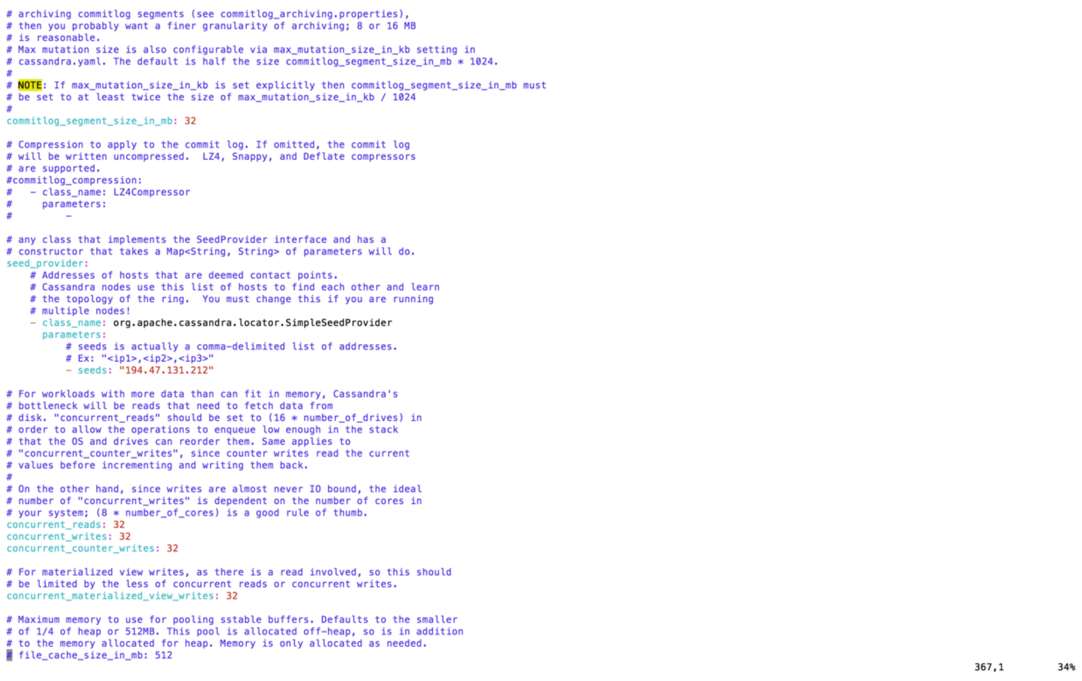
[20] “Exploring LXC Networking - Container Ops.” [Online]. Available: http://containerops.org/2013/11/19/lxc-networking/. [Accessed: 14-Sep-2016].

[21] E. Casalicchio, L. Lundberg, and S. Shirinbad, “An Energy-Aware Adaptation Model for Big Data Platforms,” in *2016 IEEE International Conference on Autonomic Computing (ICAC)*, 2016, pp. 349–350.

APPENDIX

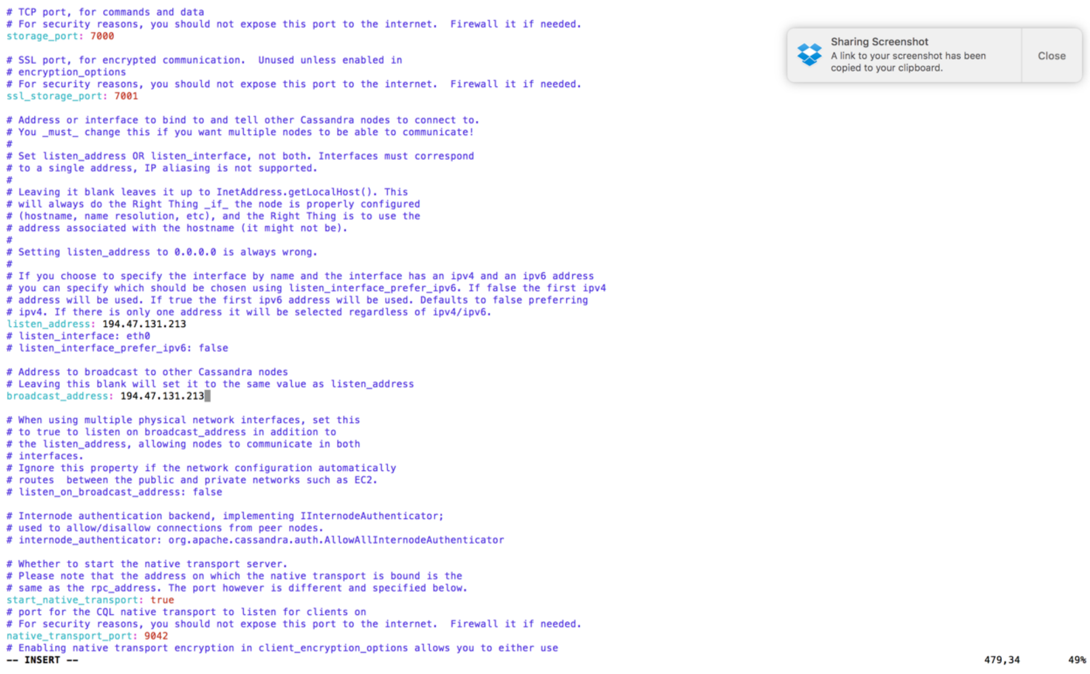
This appendix chapter provides the status and results of the experiments conducted for the purpose of Performance Comparison of Cassandra in LXC and Bare metal.

Cassandra Network Configuration



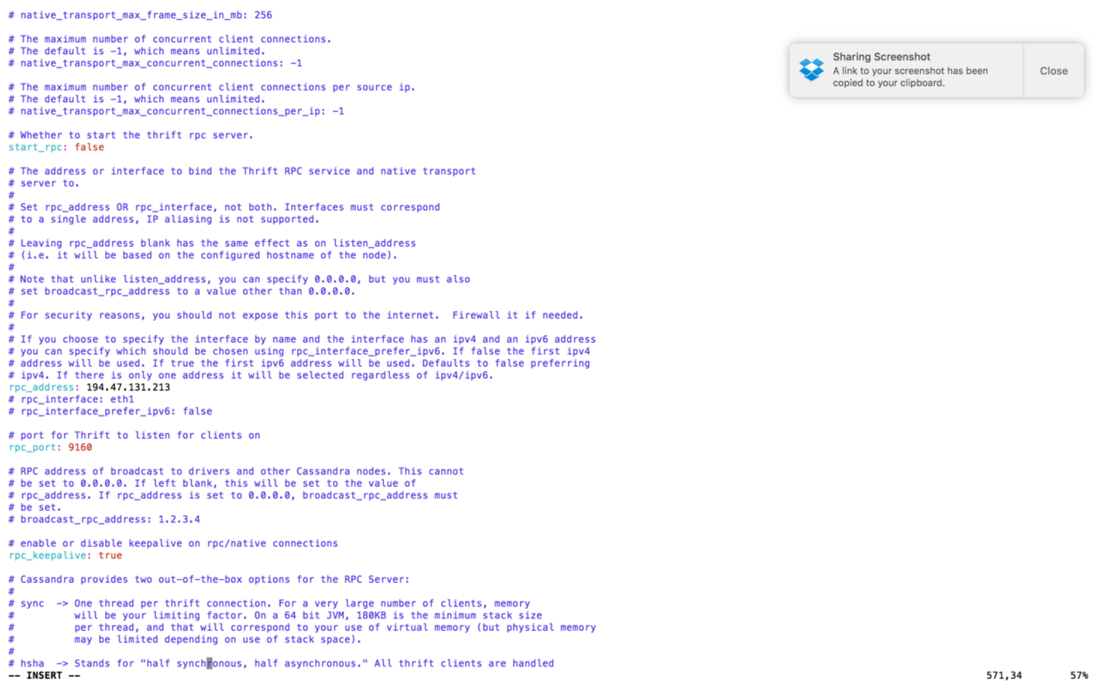
```
# archiving commitlog segments (see commitlog_archiving.properties),
# then you probably want a finer granularity of archiving; 8 or 16 MB
# is reasonable.
# Max mutation size is also configurable via max_mutation_size_in_kb setting in
# cassandra.yaml. The default is half the size commitlog_segment_size_in_mb * 1024.
#
# NOTE: If max_mutation_size_in_kb is set explicitly then commitlog_segment_size_in_mb must
# be set to at least twice the size of max_mutation_size_in_kb / 1024
#
commitlog_segment_size_in_mb: 32

# Compression to apply to the commit log. If omitted, the commit log
# will be written uncompressed.  LZ4, Snappy, and Deflate compressors
# are supported.
#commitlog_compression:
#   - class_name: LZ4Compressor
#     parameters:
#         -

# any class that implements the SeedProvider interface and has a
# constructor that takes a Map<String, String> of parameters will do.
seed_provider:
    # Addresses of hosts that are deemed contact points.
    # Cassandra nodes use this list of hosts to find each other and learn
    # the topology of the ring.  You must change this if you are running
    # multiple nodes!
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          # seeds is actually a comma-delimited list of addresses.
          # Ex: "<ip1>,<ip2>,<ip3>"
          - seeds: "194.47.131.212"

# For workloads with more data than can fit in memory, Cassandra's
# bottleneck will be reads that need to fetch data from
# disk. "concurrent_reads" should be set to (16 * number_of_drives) in
# order to allow the operations to enqueue low enough in the stack
# that the OS and drives can reorder them. Same applies to
# "concurrent_counter_writes", since counter writes read the current
# values before incrementing and writing them back.
#
# On the other hand, since writes are almost never IO bound, the ideal
# number of "concurrent_writes" is dependent on the number of cores in
# your system; (8 * number_of_cores) is a good rule of thumb.
concurrent_reads: 32
concurrent_writes: 32
concurrent_counter_writes: 32

# For materialized view writes, as there is a read involved, so this should
# be limited by the less of concurrent reads or concurrent writes.
concurrent_materialized_view_writes: 32

# Maximum memory to use for pooling sstable buffers. Defaults to the smaller
# of 1/4 of heap or 512MB. This pool is allocated off-heap, so is in addition
# to the memory allocated for heap. Memory is only allocated as needed.
# file_cache_size_in_mb: 512
                                                                                 367,1         34%
```

Figure 31 - Configuration of cassandra.yaml file (1)


Sharing Screenshot
A link to your screenshot has been copied to your clipboard.
Close

```
# TCP port, for commands and data
# For security reasons, you should not expose this port to the internet.  Firewall it if needed.
storage_port: 7000

# SSL port, for encrypted communication.  Unused unless enabled in
# encryption_options
# For security reasons, you should not expose this port to the internet.  Firewall it if needed.
ssl_storage_port: 7001

# Address or interface to bind to and tell other Cassandra nodes to connect to.
# You _must_ change this if you want multiple nodes to be able to communicate!
#
# Set listen_address OR listen_interface, not both. Interfaces must correspond
# to a single address, IP aliasing is not supported.
#
# Leaving it blank leaves it up to InetAddress.getLocalHost(). This
# will always do the Right Thing _if_ the node is properly configured
# (hostname, name resolution, etc), and the Right Thing is to use the
# address associated with the hostname (it might not be).
#
# Setting listen_address to 0.0.0.0 is always wrong.
#
# If you choose to specify the interface by name and the interface has an ipv4 and an ipv6 address
# you can specify which should be chosen using listen_interface_prefer_ipv6. If false the first ipv4
# address will be used. If true the first ipv6 address will be used. Defaults to false preferring
# ipv4. If there is only one address it will be selected regardless of ipv4/ipv6.
listen_address: 194.47.131.213
# listen_interface: eth0
# listen_interface_prefer_ipv6: false

# Address to broadcast to other Cassandra nodes
# Leaving this blank will set it to the same value as listen_address
broadcast_address: 194.47.131.213

# When using multiple physical network interfaces, set this
# to true to listen on broadcast_address in addition to
# the listen_address, allowing nodes to communicate in both
# interfaces.
# Ignore this property if the network configuration automatically
# routes  between the public and private networks such as EC2.
# listen_on_broadcast_address: false

# Internode authentication backend, implementing IInternodeAuthenticator;
# used to allow/disallow connections from peer nodes.
# internode_authenticator: org.apache.cassandra.auth.AllowAllInternodeAuthenticator

# Whether to start the native transport server.
# Please note that the address on which the native transport is bound is the
# same as the rpc_address. The port however is different and specified below.
start_native_transport: true
# port for the CQL native transport to listen for clients on
# For security reasons, you should not expose this port to the internet.  Firewall it if needed.
native_transport_port: 9042
# Enabling native transport encryption in client_encryption_options allows you to either use
-- INSERT --                                                                      479,34        49%
```

Figure 32 - Configuration of cassandra.yaml file (2)

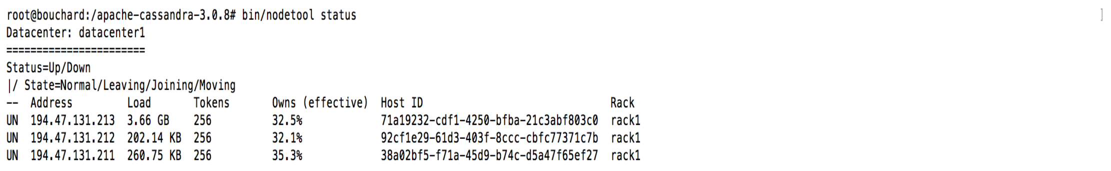


```
# native_transport_max_frame_size_in_mb: 256

# The maximum number of concurrent client connections.
# The default is -1, which means unlimited.
# native_transport_max_concurrent_connections: -1

# The maximum number of concurrent client connections per source ip.
# The default is -1, which means unlimited.
# native_transport_max_concurrent_connections_per_ip: -1

# Whether to start the thrift rpc server.
start_rpc: false

# The address or interface to bind the Thrift RPC service and native transport
# server to.
#
# Set rpc_address OR rpc_interface, not both. Interfaces must correspond
# to a single address, IP aliasing is not supported.
#
# Leaving rpc_address blank has the same effect as on listen_address
# (i.e. it will be based on the configured hostname of the node).
#
# Note that unlike listen_address, you can specify 0.0.0.0, but you must also
# set broadcast_rpc_address to a value other than 0.0.0.0.
#
# For security reasons, you should not expose this port to the internet.  Firewall it if needed.
#
# If you choose to specify the interface by name and the interface has an ipv4 and an ipv6 address
# you can specify which should be chosen using rpc_interface_prefer_ipv6. If false the first ipv4
# address will be used. If true the first ipv6 address will be used. Defaults to false preferring
# ipv4. If there is only one address it will be selected regardless of ipv4/ipv6.
rpc_address: 194.47.131.213
# rpc_interface: eth1
# rpc_interface_prefer_ipv6: false

# port for Thrift to listen for clients on
rpc_port: 9160

# RPC address of broadcast to drivers and other Cassandra nodes. This cannot
# be set to 0.0.0.0. If left blank, this will be set to the value of
# rpc_address. If rpc_address is set to 0.0.0.0, broadcast_rpc_address must
# be set.
# broadcast_rpc_address: 1.2.3.4

# enable or disable keepalive on rpc/native connections
rpc_keepalive: true

# Cassandra provides two out-of-the-box options for the RPC Server:
#
# sync  -> One thread per thrift connection. For a very large number of clients, memory
#          will be your limiting factor. On a 64 bit JVM, 180KB is the minimum stack size
#          per thread, and that will correspond to your use of virtual memory (but physical memory
#          may be limited depending on use of stack space).
#
# hsha  -> Stands for "half synchronous, half asynchronous." All thrift clients are handled
-- INSERT --                                                                                                  571,34        57%
```

Sharing Screenshot
A link to your screenshot has been copied to your clipboard.
Close

Figure 33 - Configuration of cassandra.yaml file (3)

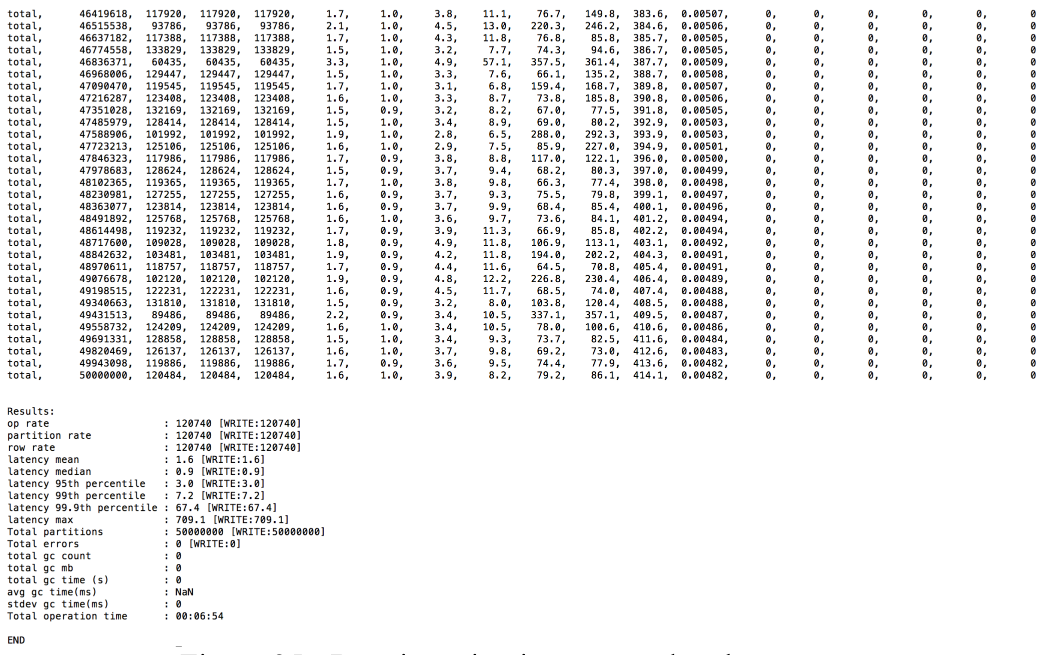
Cassandra Cluster



```
root@bouchard:/apache-cassandra-3.0.8# bin/nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
UN  194.47.131.213  3.66 GB    256          32.5%             71a19232-cdf1-4250-bfba-21c3abf803c0  rack1
UN  194.47.131.212  202.14 KB  256          32.1%             92cf1e29-61d3-403f-8ccc-cbfc77371c7b  rack1
UN  194.47.131.211  260.75 KB  256          35.3%             38a02bf5-f71a-45d9-b74c-d5a47f65ef27  rack1
```

Figure 34 - Cassandra cluster status

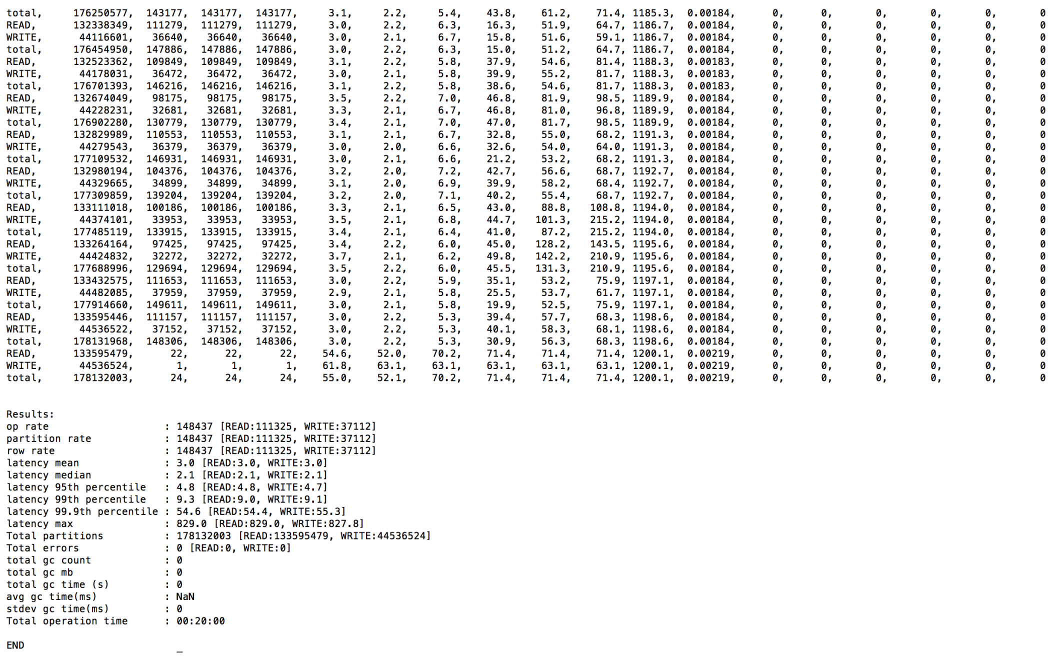
Data insertion into Cassandra cluster



```
total,      46419618, 117920, 117920, 117920,     1.7,     1.0,     3.8,    11.1,    76.7,   149.8,  383.6, 0.00507,      0,      0,      0,      0,      0,      0
total,      46515538,  93786,  93786,  93786,     2.1,     1.0,     4.5,    13.0,   220.3,   246.2,  384.6, 0.00506,      0,      0,      0,      0,      0,      0
total,      46637182, 117388, 117388, 117388,     1.7,     1.0,     4.3,    11.8,    76.8,    85.8,  385.7, 0.00505,      0,      0,      0,      0,      0,      0
total,      46774558, 133829, 133829, 133829,     1.5,     1.0,     3.2,     7.7,    74.3,    94.6,  386.7, 0.00505,      0,      0,      0,      0,      0,      0
total,      46836371,  60435,  60435,  60435,     3.3,     1.0,     4.9,    57.1,   357.5,   361.4,  387.7, 0.00509,      0,      0,      0,      0,      0,      0
total,      46968006, 129447, 129447, 129447,     1.5,     1.0,     3.3,     7.6,    66.1,   135.2,  388.7, 0.00508,      0,      0,      0,      0,      0,      0
total,      47090470, 119545, 119545, 119545,     1.7,     1.0,     3.1,     6.8,   159.4,   168.7,  389.8, 0.00507,      0,      0,      0,      0,      0,      0
total,      47216287, 123408, 123408, 123408,     1.6,     1.0,     3.3,     8.7,    73.8,   185.8,  390.8, 0.00506,      0,      0,      0,      0,      0,      0
total,      47351028, 132169, 132169, 132169,     1.5,     0.9,     3.2,     8.2,    67.0,    77.5,  391.8, 0.00505,      0,      0,      0,      0,      0,      0
total,      47485979, 128414, 128414, 128414,     1.5,     1.0,     3.4,     8.9,    69.0,    80.2,  392.9, 0.00503,      0,      0,      0,      0,      0,      0
total,      47588906, 101992, 101992, 101992,     1.9,     1.0,     2.8,     6.5,   288.0,   292.3,  393.9, 0.00503,      0,      0,      0,      0,      0,      0
total,      47723213, 125106, 125106, 125106,     1.6,     1.0,     2.9,     7.5,    85.9,   227.0,  394.9, 0.00501,      0,      0,      0,      0,      0,      0
total,      47846323, 117986, 117986, 117986,     1.7,     0.9,     3.8,     8.8,   117.0,   122.1,  396.0, 0.00500,      0,      0,      0,      0,      0,      0
total,      47978683, 128624, 128624, 128624,     1.5,     0.9,     3.7,     9.4,    68.2,    80.3,  397.0, 0.00499,      0,      0,      0,      0,      0,      0
total,      48102365, 119365, 119365, 119365,     1.7,     1.0,     3.8,     9.8,    66.3,    77.4,  398.0, 0.00498,      0,      0,      0,      0,      0,      0
total,      48230981, 127255, 127255, 127255,     1.6,     0.9,     3.7,     9.3,    75.5,    79.8,  399.1, 0.00497,      0,      0,      0,      0,      0,      0
total,      48363077, 123814, 123814, 123814,     1.6,     0.9,     3.7,     9.9,    68.4,    85.4,  400.1, 0.00496,      0,      0,      0,      0,      0,      0
total,      48491892, 125768, 125768, 125768,     1.6,     1.0,     3.6,     9.7,    73.6,    84.1,  401.2, 0.00494,      0,      0,      0,      0,      0,      0
total,      48614498, 119232, 119232, 119232,     1.7,     0.9,     3.9,    11.3,    66.9,    85.8,  402.2, 0.00494,      0,      0,      0,      0,      0,      0
total,      48717600, 109028, 109028, 109028,     1.8,     0.9,     4.9,    11.8,   106.9,   113.1,  403.1, 0.00492,      0,      0,      0,      0,      0,      0
total,      48842632, 103481, 103481, 103481,     1.9,     0.9,     4.2,    11.8,   194.0,   202.2,  404.3, 0.00491,      0,      0,      0,      0,      0,      0
total,      48970611, 118757, 118757, 118757,     1.7,     0.9,     4.4,    11.6,    64.5,    70.8,  405.4, 0.00491,      0,      0,      0,      0,      0,      0
total,      49076678, 102120, 102120, 102120,     1.9,     0.9,     4.8,    12.2,   226.8,   230.4,  406.4, 0.00489,      0,      0,      0,      0,      0,      0
total,      49198515, 122231, 122231, 122231,     1.6,     0.9,     4.5,    11.7,    68.5,    74.0,  407.4, 0.00488,      0,      0,      0,      0,      0,      0
total,      49340663, 131810, 131810, 131810,     1.5,     0.9,     3.2,     8.0,   103.8,   120.4,  408.5, 0.00488,      0,      0,      0,      0,      0,      0
total,      49431513,  89486,  89486,  89486,     2.2,     0.9,     3.4,    10.5,   337.1,   357.1,  409.5, 0.00487,      0,      0,      0,      0,      0,      0
total,      49558732, 124209, 124209, 124209,     1.6,     1.0,     3.4,    10.5,    78.0,   100.6,  410.6, 0.00486,      0,      0,      0,      0,      0,      0
total,      49691331, 128858, 128858, 128858,     1.5,     1.0,     3.4,     9.3,    73.7,    82.5,  411.6, 0.00484,      0,      0,      0,      0,      0,      0
total,      49820469, 126137, 126137, 126137,     1.6,     1.0,     3.7,     9.8,    69.2,    73.0,  412.6, 0.00483,      0,      0,      0,      0,      0,      0
total,      49943098, 119886, 119886, 119886,     1.7,     0.9,     3.6,     9.5,    74.4,    77.9,  413.6, 0.00482,      0,      0,      0,      0,      0,      0
total,      50000000, 120484, 120484, 120484,     1.6,     1.0,     3.9,     8.2,    79.2,    86.1,  414.1, 0.00482,      0,      0,      0,      0,      0,      0


Results:
op rate                   : 120740 [WRITE:120740]
partition rate            : 120740 [WRITE:120740]
row rate                  : 120740 [WRITE:120740]
latency mean              : 1.6 [WRITE:1.6]
latency median            : 0.9 [WRITE:0.9]
latency 95th percentile   : 3.0 [WRITE:3.0]
latency 99th percentile   : 7.2 [WRITE:7.2]
latency 99.9th percentile : 67.4 [WRITE:67.4]
latency max               : 709.1 [WRITE:709.1]
Total partitions          : 50000000 [WRITE:50000000]
Total errors              : 0 [WRITE:0]
total gc count            : 0
total gc mb               : 0
total gc time (s)         : 0
avg gc time(ms)           : NaN
stdev gc time(ms)         : 0
Total operation time      : 00:06:54

END
```

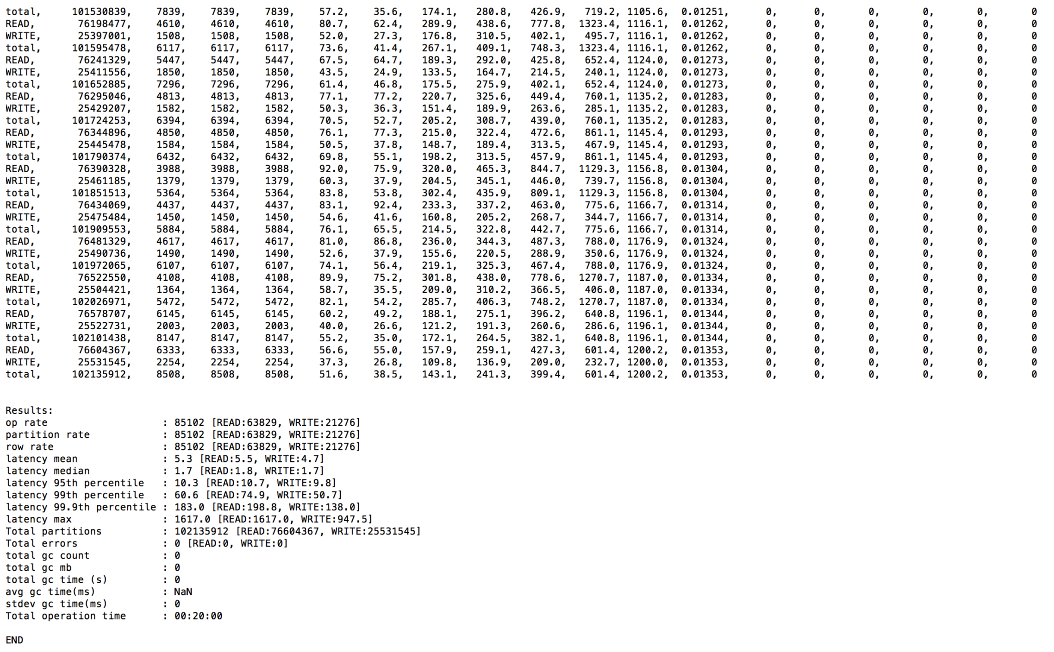
Figure 35 - Data insertion into cassandra cluster

Cassandra stress utility tool



```
total,    176250577,  143177,  143177,  143177,      3.1,      2.2,      5.4,     43.8,     61.2,     71.4,  1185.3,  0.00184,       0,       0,       0,       0,       0,       0
READ,     132338349,  111279,  111279,  111279,      3.0,      2.2,      6.3,     16.3,     51.9,     64.7,  1186.7,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44116601,   36640,   36640,   36640,      3.0,      2.1,      6.7,     15.8,     51.6,     59.1,  1186.7,  0.00184,       0,       0,       0,       0,       0,       0
total,    176454950,  147886,  147886,  147886,      3.0,      2.2,      6.3,     15.0,     51.2,     64.7,  1186.7,  0.00184,       0,       0,       0,       0,       0,       0
READ,     132523362,  109849,  109849,  109849,      3.1,      2.2,      5.8,     37.9,     54.6,     81.4,  1188.3,  0.00183,       0,       0,       0,       0,       0,       0
WRITE,     44178031,   36472,   36472,   36472,      3.0,      2.1,      5.8,     39.9,     55.2,     81.7,  1188.3,  0.00183,       0,       0,       0,       0,       0,       0
total,    176701393,  146216,  146216,  146216,      3.1,      2.2,      5.8,     38.6,     54.6,     81.7,  1188.3,  0.00183,       0,       0,       0,       0,       0,       0
READ,     132674049,   98175,   98175,   98175,      3.5,      2.2,      7.0,     46.8,     81.9,     98.5,  1189.9,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44228231,   32681,   32681,   32681,      3.3,      2.1,      6.7,     46.8,     81.0,     96.8,  1189.9,  0.00184,       0,       0,       0,       0,       0,       0
total,    176902280,  130779,  130779,  130779,      3.4,      2.1,      7.0,     47.0,     81.7,     98.5,  1189.9,  0.00184,       0,       0,       0,       0,       0,       0
READ,     132829989,  110553,  110553,  110553,      3.1,      2.1,      6.7,     32.8,     55.0,     68.2,  1191.3,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44279543,   36379,   36379,   36379,      3.0,      2.0,      6.6,     32.6,     54.0,     64.0,  1191.3,  0.00184,       0,       0,       0,       0,       0,       0
total,    177109532,  146931,  146931,  146931,      3.0,      2.1,      6.6,     21.2,     53.2,     68.2,  1191.3,  0.00184,       0,       0,       0,       0,       0,       0
READ,     132980194,  104376,  104376,  104376,      3.2,      2.0,      7.2,     42.7,     56.6,     68.7,  1192.7,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44329665,   34899,   34899,   34899,      3.1,      2.0,      6.9,     39.9,     58.2,     68.4,  1192.7,  0.00184,       0,       0,       0,       0,       0,       0
total,    177309859,  139204,  139204,  139204,      3.2,      2.0,      7.1,     40.2,     55.4,     68.7,  1192.7,  0.00184,       0,       0,       0,       0,       0,       0
READ,     133111018,  100186,  100186,  100186,      3.3,      2.1,      6.5,     43.0,     88.8,    108.8,  1194.0,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44374101,   33953,   33953,   33953,      3.5,      2.1,      6.8,     44.7,    101.3,    215.2,  1194.0,  0.00184,       0,       0,       0,       0,       0,       0
total,    177485119,  133915,  133915,  133915,      3.4,      2.1,      6.4,     41.0,     87.2,    215.2,  1194.0,  0.00184,       0,       0,       0,       0,       0,       0
READ,     133264164,   97425,   97425,   97425,      3.4,      2.2,      6.0,     45.0,    128.2,    143.5,  1195.6,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44424832,   32272,   32272,   32272,      3.7,      2.1,      6.2,     49.8,    142.2,    210.9,  1195.6,  0.00184,       0,       0,       0,       0,       0,       0
total,    177688996,  129694,  129694,  129694,      3.5,      2.2,      6.0,     45.5,    131.3,    210.9,  1195.6,  0.00184,       0,       0,       0,       0,       0,       0
READ,     133432575,  111653,  111653,  111653,      3.0,      2.2,      5.9,     35.1,     53.2,     75.9,  1197.1,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44482085,   37959,   37959,   37959,      2.9,      2.1,      5.8,     25.5,     53.7,     61.7,  1197.1,  0.00184,       0,       0,       0,       0,       0,       0
total,    177914660,  149611,  149611,  149611,      3.0,      2.1,      5.8,     19.9,     52.5,     75.9,  1197.1,  0.00184,       0,       0,       0,       0,       0,       0
READ,     133595446,  111157,  111157,  111157,      3.0,      2.2,      5.3,     39.4,     57.7,     68.3,  1198.6,  0.00184,       0,       0,       0,       0,       0,       0
WRITE,     44536522,   37152,   37152,   37152,      3.0,      2.2,      5.3,     40.1,     58.3,     68.1,  1198.6,  0.00184,       0,       0,       0,       0,       0,       0
total,    178131968,  148306,  148306,  148306,      3.0,      2.2,      5.3,     30.9,     56.3,     68.3,  1198.6,  0.00184,       0,       0,       0,       0,       0,       0
READ,     133595479,      22,      22,      22,     54.6,     52.0,     70.2,     71.4,     71.4,     71.4,  1200.1,  0.00219,       0,       0,       0,       0,       0,       0
WRITE,     44536524,       1,       1,       1,     61.8,     63.1,     63.1,     63.1,     63.1,     63.1,  1200.1,  0.00219,       0,       0,       0,       0,       0,       0
total,    178132003,      24,      24,      24,     55.0,     52.1,     70.2,     71.4,     71.4,     71.4,  1200.1,  0.00219,       0,       0,       0,       0,       0,       0


Results:
op rate                   : 148437 [READ:111325, WRITE:37112]
partition rate            : 148437 [READ:111325, WRITE:37112]
row rate                  : 148437 [READ:111325, WRITE:37112]
latency mean              : 3.0 [READ:3.0, WRITE:3.0]
latency median            : 2.1 [READ:2.1, WRITE:2.1]
latency 95th percentile   : 4.8 [READ:4.8, WRITE:4.7]
latency 99th percentile   : 9.3 [READ:9.0, WRITE:9.1]
latency 99.9th percentile : 54.6 [READ:54.4, WRITE:55.3]
latency max               : 829.0 [READ:829.0, WRITE:827.8]
Total partitions          : 178132003 [READ:133595479, WRITE:44536524]
Total errors              : 0 [READ:0, WRITE:0]
total gc count            : 0
total gc mb               : 0
total gc time (s)         : 0
avg gc time(ms)           : NaN
stdev gc time(ms)         : 0
Total operation time      : 00:20:00

END
```

Figure 36 - Cassandra stress utility tool (1)



```
total,    101530839,    7839,    7839,    7839,     57.2,     35.6,    174.1,    280.8,    426.9,    719.2,  1105.6,  0.01251,       0,       0,       0,       0,       0,       0
READ,      76198477,    4610,    4610,    4610,     80.7,     62.4,    289.9,    438.6,    777.8,   1323.4,  1116.1,  0.01262,       0,       0,       0,       0,       0,       0
WRITE,     25397001,    1508,    1508,    1508,     52.0,     27.3,    176.8,    310.5,    402.1,    495.7,  1116.1,  0.01262,       0,       0,       0,       0,       0,       0
total,    101595478,    6117,    6117,    6117,     73.6,     41.4,    267.1,    409.1,    748.3,   1323.4,  1116.1,  0.01262,       0,       0,       0,       0,       0,       0
READ,      76241329,    5447,    5447,    5447,     67.5,     64.7,    189.3,    292.0,    425.8,    652.4,  1124.0,  0.01273,       0,       0,       0,       0,       0,       0
WRITE,     25411556,    1850,    1850,    1850,     43.5,     24.9,    133.5,    164.7,    214.5,    240.1,  1124.0,  0.01273,       0,       0,       0,       0,       0,       0
total,    101652885,    7296,    7296,    7296,     61.4,     46.8,    175.5,    275.9,    402.1,    652.4,  1124.0,  0.01273,       0,       0,       0,       0,       0,       0
READ,      76295046,    4813,    4813,    4813,     77.1,     77.2,    220.7,    325.6,    449.4,    760.1,  1135.2,  0.01283,       0,       0,       0,       0,       0,       0
WRITE,     25429207,    1582,    1582,    1582,     50.3,     36.3,    151.4,    189.9,    263.6,    285.1,  1135.2,  0.01283,       0,       0,       0,       0,       0,       0
total,    101724253,    6394,    6394,    6394,     70.5,     52.7,    205.2,    308.7,    439.0,    760.1,  1135.2,  0.01283,       0,       0,       0,       0,       0,       0
READ,      76344896,    4850,    4850,    4850,     76.1,     77.3,    215.0,    322.4,    472.6,    861.1,  1145.4,  0.01293,       0,       0,       0,       0,       0,       0
WRITE,     25445478,    1584,    1584,    1584,     50.5,     37.8,    148.7,    189.4,    313.5,    467.9,  1145.4,  0.01293,       0,       0,       0,       0,       0,       0
total,    101790374,    6432,    6432,    6432,     69.8,     55.1,    198.2,    313.5,    457.9,    861.1,  1145.4,  0.01293,       0,       0,       0,       0,       0,       0
READ,      76390328,    3988,    3988,    3988,     92.0,     75.9,    320.0,    465.3,    844.7,   1129.3,  1156.8,  0.01304,       0,       0,       0,       0,       0,       0
WRITE,     25461185,    1379,    1379,    1379,     60.3,     37.9,    204.5,    345.1,    446.0,    739.7,  1156.8,  0.01304,       0,       0,       0,       0,       0,       0
total,    101851513,    5364,    5364,    5364,     83.8,     53.8,    302.4,    435.9,    809.1,   1129.3,  1156.8,  0.01304,       0,       0,       0,       0,       0,       0
READ,      76434069,    4437,    4437,    4437,     83.1,     92.4,    233.3,    337.2,    463.0,    775.6,  1166.7,  0.01314,       0,       0,       0,       0,       0,       0
WRITE,     25475484,    1450,    1450,    1450,     54.6,     41.6,    160.8,    205.2,    268.7,    344.7,  1166.7,  0.01314,       0,       0,       0,       0,       0,       0
total,    101909553,    5884,    5884,    5884,     76.1,     65.5,    214.5,    322.8,    442.7,    775.6,  1166.7,  0.01314,       0,       0,       0,       0,       0,       0
READ,      76481329,    4617,    4617,    4617,     81.0,     86.8,    236.0,    344.3,    487.3,    788.0,  1176.9,  0.01324,       0,       0,       0,       0,       0,       0
WRITE,     25490736,    1490,    1490,    1490,     52.6,     37.9,    155.6,    220.5,    288.9,    350.6,  1176.9,  0.01324,       0,       0,       0,       0,       0,       0
total,    101972065,    6107,    6107,    6107,     74.1,     56.4,    219.1,    325.3,    467.4,    788.0,  1176.9,  0.01324,       0,       0,       0,       0,       0,       0
READ,      76522550,    4108,    4108,    4108,     89.9,     75.2,    301.8,    438.0,    778.6,   1270.7,  1187.0,  0.01334,       0,       0,       0,       0,       0,       0
WRITE,     25504421,    1364,    1364,    1364,     58.7,     35.5,    209.0,    310.2,    366.5,    406.0,  1187.0,  0.01334,       0,       0,       0,       0,       0,       0
total,    102026971,    5472,    5472,    5472,     82.1,     54.2,    285.7,    406.3,    748.2,   1270.7,  1187.0,  0.01334,       0,       0,       0,       0,       0,       0
READ,      76578707,    6145,    6145,    6145,     60.2,     49.2,    188.1,    275.1,    396.2,    640.8,  1196.1,  0.01344,       0,       0,       0,       0,       0,       0
WRITE,     25522731,    2003,    2003,    2003,     40.0,     26.6,    121.2,    191.3,    260.6,    286.6,  1196.1,  0.01344,       0,       0,       0,       0,       0,       0
total,    102101438,    8147,    8147,    8147,     55.2,     35.0,    172.1,    264.5,    382.1,    640.8,  1196.1,  0.01344,       0,       0,       0,       0,       0,       0
READ,      76604367,    6333,    6333,    6333,     56.6,     55.0,    157.9,    259.1,    427.3,    601.4,  1200.2,  0.01353,       0,       0,       0,       0,       0,       0
WRITE,     25531545,    2254,    2254,    2254,     37.3,     26.8,    109.8,    136.9,    209.0,    232.7,  1200.0,  0.01353,       0,       0,       0,       0,       0,       0
total,    102135912,    8508,    8508,    8508,     51.6,     38.5,    143.1,    241.3,    399.4,    601.4,  1200.2,  0.01353,       0,       0,       0,       0,       0,       0


Results:
op rate                   : 85102 [READ:63829, WRITE:21276]
partition rate            : 85102 [READ:63829, WRITE:21276]
row rate                  : 85102 [READ:63829, WRITE:21276]
latency mean              : 5.3 [READ:5.5, WRITE:4.7]
latency median            : 1.7 [READ:1.8, WRITE:1.7]
latency 95th percentile   : 10.3 [READ:10.7, WRITE:9.8]
latency 99th percentile   : 60.6 [READ:74.9, WRITE:50.7]
latency 99.9th percentile : 183.0 [READ:198.8, WRITE:138.0]
latency max               : 1617.0 [READ:1617.0, WRITE:947.5]
Total partitions          : 102135912 [READ:76604367, WRITE:25531545]
Total errors              : 0 [READ:0, WRITE:0]
total gc count            : 0
total gc mb               : 0
total gc time (s)         : 0
avg gc time(ms)           : NaN
stdev gc time(ms)         : 0
Total operation time      : 00:20:00

END
```

Figure 37 - Cassandra stress utility tool (2)

For 66% load

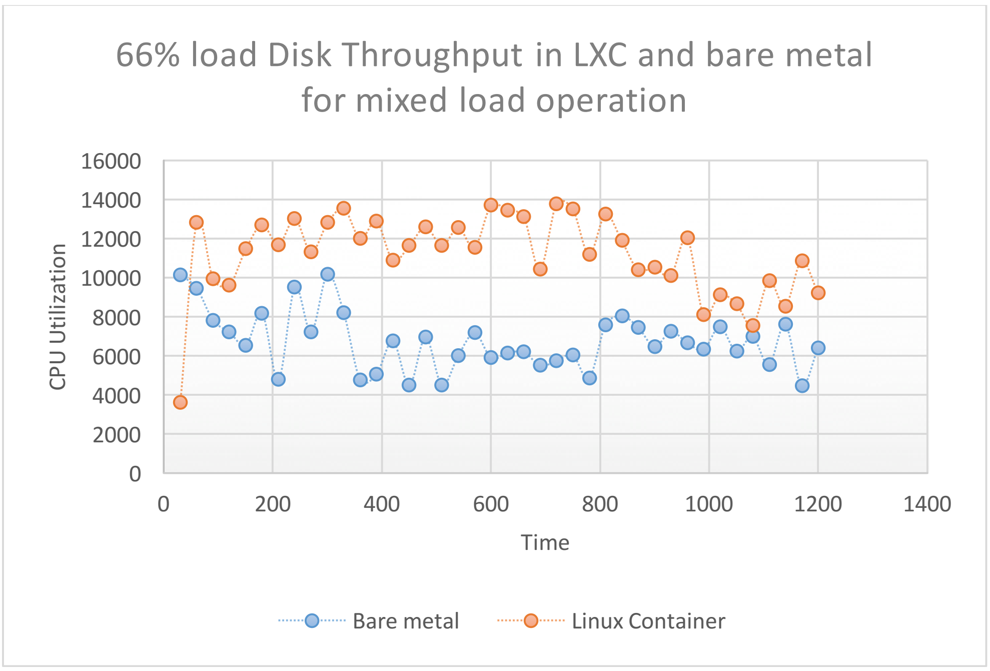


Figure 38 - 66% load Disk throughput in LXC and bare metal for mixed load operation

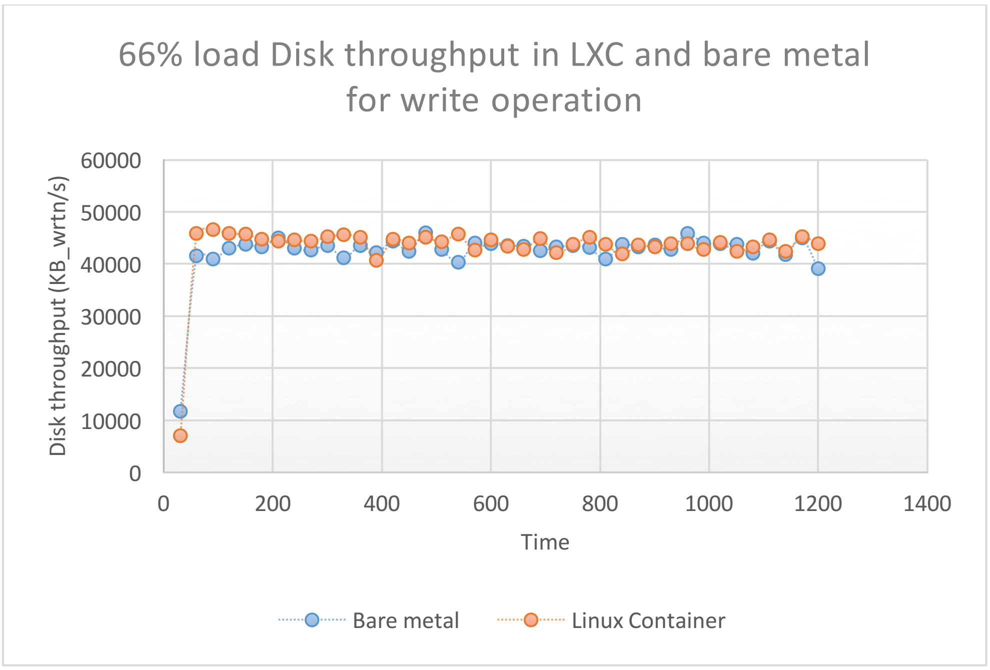


Figure 39 - 66% load Disk throughput in LXC and bare metal for write operation