Performance Evaluation of Containers and Virtual Machines when Running Cassandra Workload Concurrently

Sogand Shirinbab, Lars Lundberg, Emiliano Casalicchio
Department of Computer Science
Blekinge Institute of Technology
Karlskrona, Sweden

{Sogand.Shirinbab, Lars.Lundberg, Emiliano.Casalicchio}@bth.se

***Abstract* —** NoSQL distributed databases are largely used as Big Data platforms. To provide efficient resource sharing and cost effectiveness, such distributed databases typically run concurrently on a virtualized infrastructure that could be implemented using hypervisor-based virtualization or container-based virtualization. Hypervisor-based virtualization is a mature technology but imposes overhead on CPU, networking, and disk. Recently, by sharing the operating system resources and simplifying the deployment of applications, container-based virtualization is getting more popular.

This paper presents a performance comparison between multiple instances of VMware VMs and Docker containers running concurrently. The workload is Apache Cassandra, which is a NoSQL distributed database for Big Data platforms. As a baseline, we evaluated the performance of Cassandra when running on the non-virtualized physical infrastructure. Our study shows that Docker had lower overhead compared to VMware; the Cassandra performance on the container-based infrastructure was as good as on the non-virtualized. Our performance evaluations also show that the running multiple instances of a Cassandra database concurrently affected the performance of read and write operations differently; for both VMware and Docker, the maximum number of read operations was reduced when we ran several instances concurrently, whereas the maximum number of write operations increased when we ran instances concurrently.

Index Terms — ***Cassandra, Cloud computing, Containers, Virtual machine, Performance evaluation.***

1. INTRODUCTION

Hypervisor-based virtualization began several decades ago and since then it has been widely used in Cloud Computing. Hypervisors, also called Virtual Machine Monitors (VMM), share the hardware resources of a real machine between multiple Virtual Machines (VMs). By virtualizing system resources such as CPUs, memory and interrupts, it became possible to run multiple Operating Systems (OS) concurrently. The most commonly used hypervisors are Kernel Virtual Machine (KVM), Xen Server, VMware and Hyper-V. Hypervisor-based virtualization enables new features such as performance management, elastic resource scaling, and reliability services to be applied without requiring modifications to applications or operating systems. It also enables virtual machine migration for load balancing to eliminate hotspots and consolidation to improve resource utilization and energy efficiency. However, hypervisor level virtualization introduces performance overheads as studied in [4][5][6][7][8] and still limits it from being used in performance critical domains [1][2][3].

Recently, container-based virtualization has gained more popularity than hypervisor-based virtualization. A container is a light weight operating system running inside the host system. An application running in a container has unshared access to a copy of the operating system. In other words, containers virtualize the operating system while hypervisors virtualize the hardware resources. Therefore, container-based virtualization is well-known for providing savings in resource consumption without the overhead of hypervisor-based virtualization while also providing isolation [9]. The main difference between the VM and the container architecture is that, for the VMs, each virtualized application includes an entire guest operating system and necessary binaries/libraries, while for the containers, the container engine contains just the application and its dependencies (binaries/libraries).

Container-based virtualization is not a new concept; it has been offered before by FreeBSD Jails (available since 2000) and Solaris Zones (available since 2004). In the beginning of 2008 a new Linux kernel was released replacing the earlier variations in the form of Linux container (LXC) [10][11][12]. Other alternatives to Linux-based containers are Open VZ [13][14][15] and Docker [16][17][18]. Recently, there have been several studies on performance of container-based virtualization technologies, especially Docker containers [19][20]. Docker containers are designed to run a single application per container while LXC containers are more like virtual machines with a fully functional operating system [16]. The container-based architecture is rapidly becoming a popular development and deployment paradigm because of advantages such as low overhead and portability. Disadvantages with using containers are: security issues (which will not be the focus in this study [21]); the limitations of not being able to run a different OS; and that the maturity level of management tools (e.g., for live migration, snapshotting and resizing) is lower than for VMs.

Since both containers and VMs have their set of benefits and drawbacks, one of the key points to select the proper virtualization technology for big data platforms is to assess the performance of hypervisor-based virtualized or container-based virtualized databases and how this relates to physical ones [22]. This paper provides a detailed performance comparison running a distributed database on a number of physical servers, using VMware VMs, or using Docker containers. As database we selected Apache Cassandra [23][24][25] an open-source NoSQL distributed database widely adopted by companies using Docker and VMware for production and widely used in Big Data applications. Cassandra is known to manage some of the world's largest datasets on clusters with many thousands of nodes deployed across multiple data centers. Also, Cassandra Query Language (CQL) is user-friendly and declarative [26][27].

Our experimental results show that the container-based version had lower overhead compared to the VMware virtualized version. In fact, the performance of the container-based version was as good as non-virtualized version.

The presented work is organized as follows: In Section 2 we discuss related work. Section 3 describes the experimental setup and test cases. Section 4 presents the experimental results and we conclude our work in Section 5.

2. Related Work

Today, the container virtualization technology is gaining momentum [29][30][31]. While hypervisor-based virtualization (i.e., virtual machine) is a mature technology [4][5][6][7][8], which was introduced by IBM mainframes in the 1970s [28], container-based technologies (e.g., Docker and Linux Container (LXC)) have been introduced recently [51]. The major advantage of containers is that they achieve near-native performance. This characteristic makes it reasonable to deploy containers on top of VMs or directly on bare-metal. The latter, is an option interesting for specific settings, like public and private clouds or enterprise distributed applications; running containers on VMs solve isolation security issues in container-based applications.

This research focuses on distributed databases for big data applications. However, the use of containers is investigated also in different contexts. For that reason, in what follow we classify the related work in studies focused on distributed databases workload, and on studies addressing other types of concurrent workloads.

2.1 Distributed databases workload

Databases are often chosen to store and query large amounts of data. Traditionally, SQL databases were used in most data centers. However, because of scalability issues, NoSQL databases have gained popularity since 2007 [36]. NoSQL databases support large volumes of data in a scalable and flexible manner. Currently, in the majority of data centers, online applications with NoSQL databases are hosted on virtual machines. Therefore, there is a need to evaluate NoSQL database performance in virtual environments. There are various studies trying to assess the performance impacts of virtualization on SQL and NoSQL databases.

In [37] the authors compared the performance of three databases: a SQL database (PostgreSQL), and two NoSQL databases (MongoDB and Cassandra) using sensor data. They also compared running these databases on a physical machine and on a virtual machine. According to their results, virtualization has a huge effect on Cassandra read performance while it has a moderate performance impact on MongoDB, and increase the write performance on PostgreSQL. The difference between their work and our work is that, we were more interested in comparison between performance impacts of container-based virtualization and hypervisor-based virtualization on the Cassandra database. Another difference is that they have used only one machine to run their experiment while in our work we have used multiple machines. In [38] the authors compared the performance of Docker container and KVM hypervisor using the MySQL database. According to their results, containers show equal or better performance than virtual machines in almost all cases. Their work is similar to our work. However, we chose to evaluate Cassandra because it is a popular NoSQL database and it is widely used in the cloud. A similar study is provided in [41]. In [39] the authors compared the performance of two NoSQL databases (MongoDB and Cassandra) using different virtualization techniques, VMware (full virtualization) and Xen (paravirtualization). According to their results VMware provides better resource utilization for NoSQL databases. Their work is different from our work; in our study we considered performance of Cassandra on VMware, and container-based virtualization using Docker.

2.2 Other concurrent workloads

There are a number of studies comparing hypervisor-based and container-based virtualization.

Measuring container performance is still under discussion. In [42] the authors assess the different measurement methodology for CPU and disk I/O intensive Docker workloads. While in [44] the authors presented a preliminary study that correlates performance counters from the `/cgroup` file system with performance counters from the `/proc` file system.

In [32] the authors compared the performance of the KVM and Xen hypervisors with Docker containers on the ARM architecture. According to their results, containers had better performance in CPU bound workloads and request/response networking, while hypervisors had better performance in disk I/O operations (because of the hypervisor's caching mechanisms) and TCP streaming benchmarks. Similarly, in [33] and [34], the authors compared the performance of container-based virtualization with hypervisor-based virtualization for HPC. According to their result, container-based solutions delivered better performance than hypervisor-based solutions. Our work has some similarities to these works.

In [35] the authors characterized the performance of three common hypervisors: KVM, Xen, VMware ESXi and an open source container LXC across two generations of GPUs and two host microarchitectures, and across three sets of benchmarks. According to their results KVM achieved 98-100% of the base system's performance while VMware and Xen achieved only 96-99%. In [13] the authors compared the performance of an open source container technology OpenVZ and the Xen hypervisor. According to their results, container-based virtualization outperforms hypervisor-based virtualization. Their work is similar to our study. However, they considered Wide-Area Motion Imagery (WAMI), Full Motion Video (FMV), and text data, while in our case study we were more interested in the performance of the Cassandra database.

In [2] the authors compared the execution times of AutoDock3 (a scientific application) for Docker containers and VMs created using OpenStack. According to their results, the overall execution times for

container-based virtualization systems are less than in hypervisor-based virtualization systems due to differences in start-up times. In [9] the authors analyzed the process handling, file systems and namespace isolation for container-based virtualization systems such as Docker, Linux Containers (LXC), OpenVZ and Warden. From their assessment, containers have an advantage over VMs because of performance improvements and reduced start-up times. In [3] the authors demonstrated that container-based systems are more suitable for usage scenarios that require high levels of isolation and efficiency such as HPC clusters. Their results indicate that container-based systems perform two times better for server-type workloads than hypervisor-based systems.

In [45] the authors studied the performance of container platforms running on top the NeCTAR cloud infrastructure. Specifically, the authors compared the performance of Docker, Flockport (LXC) and VMs using the same benchmarks as in [38]. The comparison was intended to explore the performance of CPU, memory, network and disk.

In [46] the authors proposed a study on the interference among multiple applications sharing the same resources and running in Docker containers. The study focuses on I/O and it proposes also a modification of the Docker kernel to collect the maximum I/O bandwidth from the machine it is running on.

The performance of containers running on Internet-of-Things Raspberry Pi devices are investigated in [47]. As benchmarks are used: system benchmarks to independently stress CPU, memory, network I/O, disk I/O; and application benchmarks reproducing MySQL and Apache workload. The point of reference for comparison is the performance of the system without any virtualization.

In [48] the authors investigate the impact of Docker configuration and resource interference on the performance and scalability of Spark based big data applications.

In [49] the authors present an extensive study of Docker storage, for wide range of file systems, and demonstrate its impact on system and workload performance. In [50] the authors compared the performance of Docker when the Union file system and the CoW file system are used to build image layers.

3. EVALUATION

The goal of the experiment was that of comparing the performance of VMware virtual machines and Docker containers when running Cassandra. As one baseline for the comparison we used the performance of Cassandra running on physical servers without any virtualization.

3.1 Experimental Setup

All our tests were performed on three HP servers DL380 G7 with a total of 16 processor cores (plus HyperThreading), 64 GB of RAM and disk of size 400 GB. Cassandra 3.0.8 run on RHEL7 and this setup was identical for all test cases. The same version of Cassandra was used as well as for the load generators. VMware ESXi 6.0.0 was installed in case of virtualized workload using VMs. We used Docker version 1.12.6 in our container measurements. In our experiments we consider the cases of 1, 2, and 4 Cassandra clusters concurrently running on the test-bed. Each Cassandra cluster consists of three Cassandra (virtual) nodes. Each virtual node is either implemented as a VM or as a container, and each of the three virtual nodes in a Cassandra cluster runs on different physical servers (there are three physical servers). Default settings are used to configure Cassandra and only the Replication Factor parameter is changed in the experiments.

3.2 Workload

To generate workload, we used the Cassandra-stress tool. The Cassandra-stress tool is a Java-based stress utility for basic benchmarking and load testing of a Cassandra cluster. Creating the best data model requires significant load testing and multiple iterations. The Cassandra-stress tool helps us by populating our cluster and supporting stress testing of arbitrary CQL tables and arbitrary queries on tables. The Cassandra package comes with a command-line stress tool (Cassandra-stress tool) to generate load on the cluster of servers, the

`cqlsh` utility, a python-based command line client for executing Cassandra Query Language (CQL) commands and the `nodetool` utility for managing a cluster. These tools are used to stress the servers from the client and manage the data in the servers.

The Cassandra-stress tool creates a keyspace called keyspace1 and within that, tables named standard1 or counter1 in each of the nodes. These are automatically created the first time we run the stress test and are reused on subsequent runs unless we drop the keyspace using CQL. A write operation inserts data into the database and is executed prior to the load testing of the database. Later, after the data are inserted into the database, we run the mix workload, and then split up the mix workload and run a write only workload and a read only workload.

TABLE I. CASSANDRA-STRESS TOOL SAMPLE COMMANDS

	Command
Populate the Database	`cassandra-stress write n=40000000 -pop seq=1..40000000 -node -schema "replication(strategy=NetworkTopologyStrategy, datacenter1=3)"`
Mix-Load	`cassandra-stress mixed ratio\(write=1, read=3\) duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node`
Read-Load	`cassandra-stress read duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node`
Write-Load	`cassandra-stress write duration=30m -pop seq=1..40000000 -schema keyspace="keyspace1" -rate threads=100 limit=op/s –node`

Below we described in detail each workload as well as the commands we used for generating the workloads:

- *Mix-Load:* To analyze the operation of a database while running both read and write operations during one operation, a mixed load command is used to populate the cluster. A mixed load operation consists of 75% read requests and 25% write requests (three reads and one write) generated for a duration of 30 minutes onto a three-node Cassandra cluster. The command used for generating the mixed load is described in Table I. In case of 2 or 4 Cassandra clusters, we have a separate workload generator for each Cassandra cluster.
- *Read-Load:* In addition to the mix workload, we measured the performance of the database for a read-only workload. In this case, a read load command is used to populate the cluster. A read load operation consists of 100% read requests generated for a duration of 30 minutes onto a three-node Cassandra cluster. The command used for generating the read-load is described in Table I. In case of 2 or 4 Cassandra clusters, we have a separate workload generator for each cluster.
- *Write-Load:* In addition to the mix and read workload, we measured the performance of the database for the write-only workload. In this case, a write load command is used to populate the cluster. A write load operation consists of 100% write requests generated for duration of 30 minutes onto the three-node Cassandra cluster. The command used for generating the write-load is described in Table I. In case of 2 or 4 Cassandra clusters, we have a separate workload generator for each cluster.

Short descriptions of the Cassandra-stress tools input parameters (used in Table I) is provided below:

- `mixed`: Interleave basic commands with configurable ratio and distribution. The cluster must first be populated by a write test. Here we selected a mixed load operation of 75% reads and 25% write.
- `write`: Multiple concurrent writes against the cluster.
- `read`: Multiple concurrent reads against the cluster.
- `n`: Specify the number of operations to run. Here we chose n = 40000000 to generate a 10 GB database.

- `pop`: Population distribution and intra-partition visit order. In this case we chose *seq* = 1…40000000. This ensures that generated values do not overlap.
- `node`: To specify the address of the node to which data is to be populated.
- `schema`: Replication settings, compression, compaction, and so on. Here for the write operation we have modified the replication strategy to "NetworkTopologyStrategy" and set the number of replication from 1 to 3. Later for the mixed load we just set the name of the default keyspace which is "keyspace1".
- `duration`: It specifies the time in minutes to run the load.
- `rate`: Thread count, rate limit, or automatic mode (default is auto). In order to control the incoming traffic, we set the number of threads to 100. We have also limited the number of operations per second, so that we can measure the CPU utilization, write rate and the latency mean for different number of Transactions Per Seconds (tps) (40K, 80K, 120K, 160K, and 200K tps).

3.3 Number of Cassandra clusters

There are three Cassandra nodes in each Cassandra cluster; one node on each physical machine. In case of two Cassandra clusters, there are two Cassandra nodes (belonging to different clusters) on each physical machine, and in case of four Cassandra clusters, there are four Cassandra nodes (belonging to different clusters) on each physical machine. Each Cassandra node is implemented as a VMware VM or as a Docker container, depending on the selected virtualization technology.

3.4 Performance metrics

The performance of Docker, VMware, and the non-virtualized solutions are measured using the following metrics:

- CPU utilization, and
- mean latency.

The CPU utilization is measured directly on the server nodes using `sar`. The latency is measured on the client side, by the stress test tool.

3.5 Test cases

As mentioned before, three different deployment of Cassandra clusters are considered:

- *Cassandra-Non-Virtualized*: In this case, three servers are allocated and on each server we run a Cassandra node. All servers are connected using a high speed isolated LAN and create a three-node Cassandra cluster.
- *Cassandra-VM*: In this case, one VMware virtual machine per Cassandra node is instantiated on each host, i.e., for the case with four Cassandra clusters there are in total 12 VMware VMs. Each Cassandra cluster is spread out on the three physical servers.
- *Cassandra-Docker*: In this case, a version of Cassandra using Docker container is created and one container per Cassandra node is deployed on each server, i.e., for the case with four Cassandra clusters there are in total 12 Docker containers. Each Cassandra cluster is spread out on the three physical servers.

In each test case, we first experiment with Cassandra different workload scenarios, i.e., the Mix, Read and Write workloads. Second, for the case with one Cassandra cluster, we experiment with various Replication-Factor configurations for the Cassandra cluster. In Cassandra, the input splits (i.e., a set of table rows) are

replicated among the nodes based on a user-set Replication-Factor (RF). This design prevents data loss and helps with fault tolerance in case of node failure [40]. In our experiments, we investigate three different replication-factor settings: RF = 1, RF = 2, and RF = 3. In our test environment with three Cassandra node clusters, replication factor three means that each node should have a copy of the input data splits. For the case of RF = 2, the seed node decides, usually based on a random allocation algorithm, where to store replicas (since Cassandra v3.0 different algorithms can be selected [52]). In the case of RF = 1, each node only receives a portion of the input data (i.e., of the table rows) and no replicas are created. In the case of 2 or 4 Cassandra clusters we show results only for replication factor RF = 3, that is the most critical case, as explained in what follows.

4. Experimental Results

Figure 1 gives an overview of our results. The figure shows the maximum number of transactions per second on the same hardware for different numbers of Cassandra clusters implemented as VMs or containers, given a replication factor RF = 3. The figure shows that the performance of Docker containers is in general 30% higher than the performance of VMs, and, as expected, that the performance for the read work load is higher than the performance for the write work load. However, the figure shows that the effect of having multiple Cassandra clusters is different for the write and the read work load; writing benefits from having multiple Cassandra clusters, whereas the read performance is highest for one cluster. There are two effects that affect the performance in the case of multiple Cassandra clusters: first the overhead increases when the number of clusters increases for both the container and the VM case (see Figures 4 – 6 for details); this is the reason why the performance for the read work load decreases when the number of clusters increases. Second, the VMware hypervisor and the Docker system use affinity scheduling in the sense that a VM or a container tends to be scheduled on a certain subset of the processor cores. As a consequence, the cache hit ratio increases in case of multiple Cassandra clusters. For the read work load, data can be freely copied to all processor caches, so affinity scheduling has no, or little, effect on the cache hit ratio. For the write work load, the copies on other caches need to be invalidated when a data item is written, i.e., for this case the increased cache hit ratio due to affinity scheduling becomes important. This is the reason why the performance for the write work load increases when the number of clusters increases.

In the rest of this section we will look at more detailed results.

4.1 One Cassandra Cluster

Figure 2 shows the results of running one three-node Cassandra cluster in a non-virtualized environment, a VM environment, or a container environment while having a mix load. The results show that both the non-virtualized and the container environments with RF = 1 can handle a maximum of 200K (tps). The VM environment with RF = 1 can only handle 160K (tps). For the VM case the CPU overhead is up to 29% higher compared to the non-virtualized version. One reason for getting very high overhead could be the additional layers of processing required by the virtualization software, i.e., VMware [41]. Moreover, the container case work completely in memory, which makes it possible to cope with the small overhead introduced by containers and to reach the same performance as the non-virtualized case. This feature of containers is fully exploited in the case of RF = 2 and RF = 3 when the container version significantly outperforms the VM version and has a throughput equal or higher than the non-virtualized case.

The effect of full in-memory access to the data is evident also if we observe the latency. For RF = 1 the non-virtualized and the container versions have the same latency, which is much lower than the latency for the VM case. For RF = 2, the container case can serve a higher number of transaction than the other two cases; and for RF = 3, while the number of transactions per second is the same for the non-virtualized and the container cases, the latter shows a somewhat lower latency.

In general, RF = 1 is always performing better than RF = 2 and RF = 3. One reason is that when the RF in set to one, only one copy of data is written and this process is very fast in Cassandra. Figure 3 shows that the

latency mean for RF = 1 is lower than for RF = 2 and RF = 3. Although RF = 1 is fast, from a high availability point of view the data center providers prefer RF = 2 or RF = 3. Cassandra is using different consistency levels in order to wait for the respond from several nodes, and in our case we set the consistency level to be Quorum. Quorum means that Cassandra returns the record after a quorum of replicas have responded.

Table II shows a summary of the CPU utilizations and mean latencies for the case with one Cassandra cluster. The table shows that for all workloads and replication factors the CPU utilization and the latency were lowest for the non-virtualized case, and that VMware has the largest CPU utilization and also the highest latency.

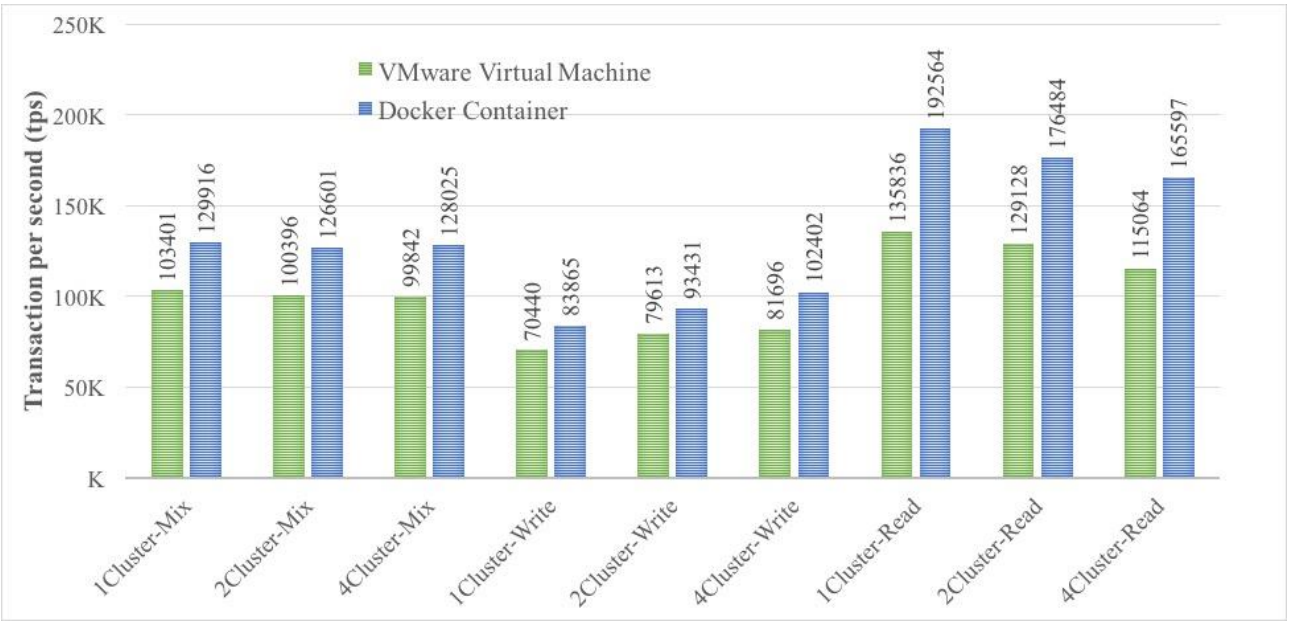

Figure 1: Transactions per second for VMware and Docker for different workloads and for different number of clusters (RF = 3).

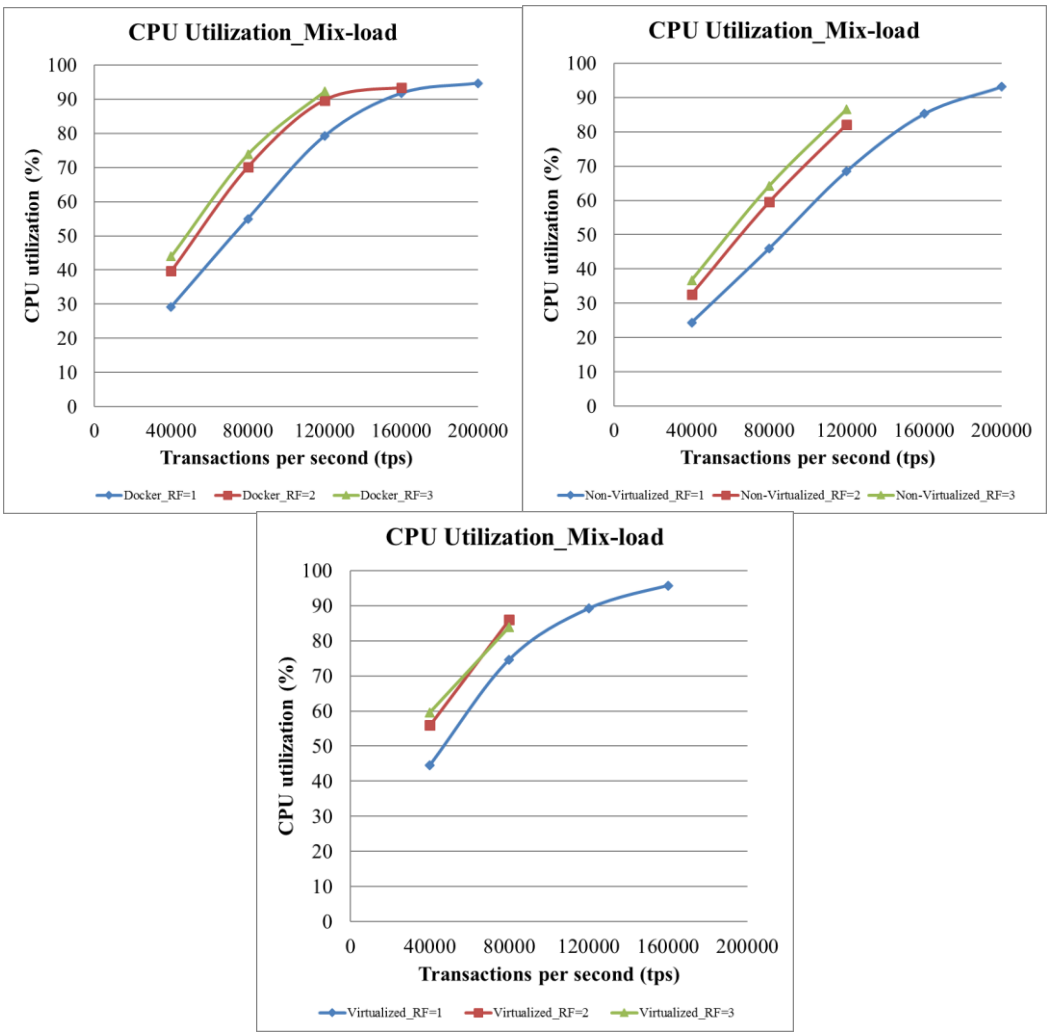

Figure 2: CPU utilization for the mixed work load for Docker (upper left), VMware (lower) and non-virtulized (upper right).

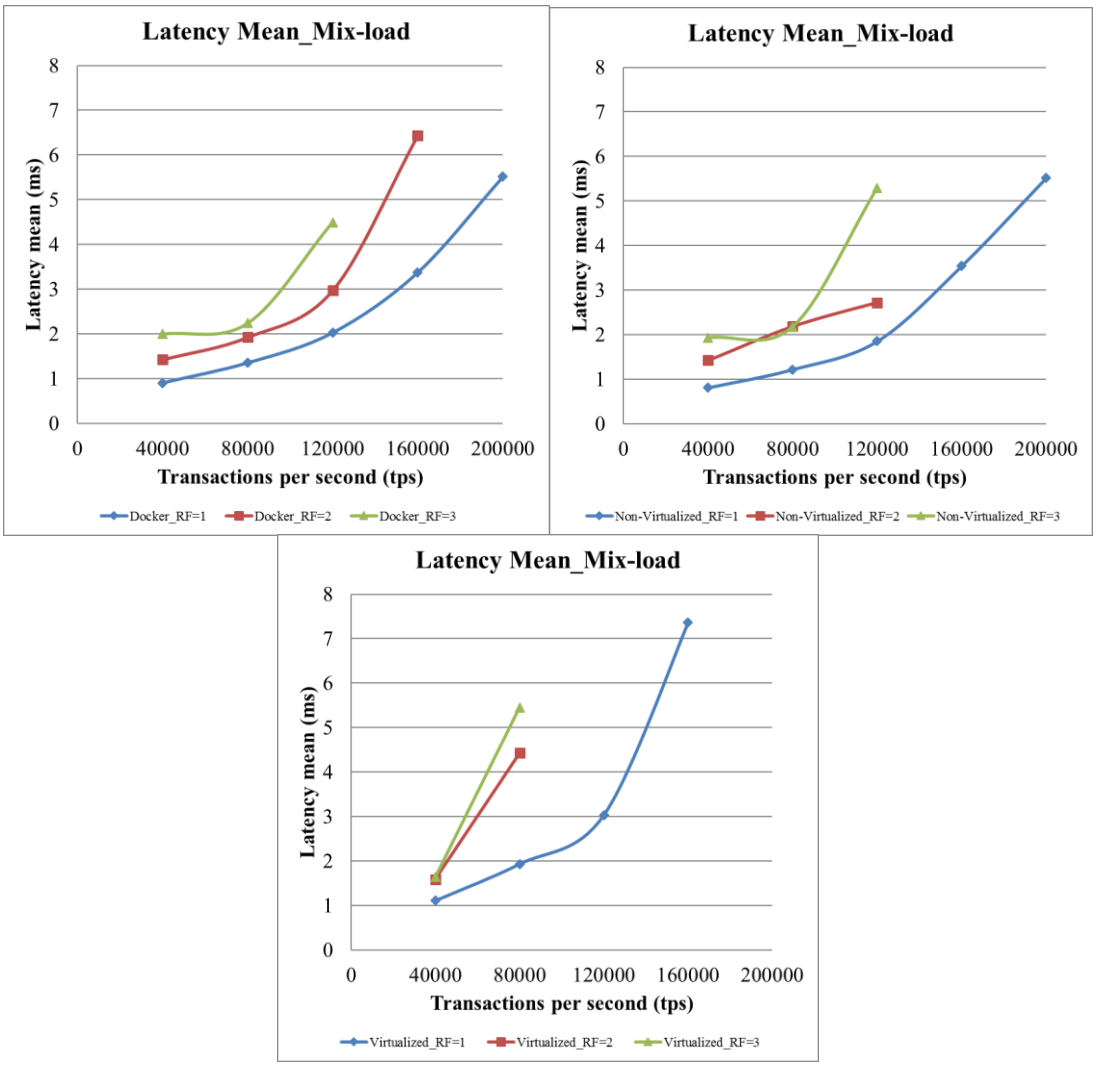

Figure 3: Mean latency for the mixed work load for Docker (upper left), VMware (lower) and non-virtulized (upper right).

TABLE II. SUMMARY OF THE PERFORMANCE RESULTS FOR ONE CASSANDRA CLUSTER

RF	Workload	Docker		Non-Virtualized		VMware	
		CPU Util (%)	*Latency (msec)*	*CPU Util (%)*	*Latency (msec)*	*CPU Util (%)*	*Latency (msec)*
1	***Mix (80K tps)***	55	1.35	46	1.2	75	1.9
	Read (60k tps)	39	0.9	32	0.9	57	1.2
	Write (20k tps)	14	0.7	12	0.6	24	0.8
2	***Mix (80k tps)***	70	1.9	59	2.1	86	4.4
	Read (60k tps)	50	1.5	41	1.4	65	1.6
	Write (20k tps)	22	0.7	18	0.7	36	0.9
3	***Mix (80k tps)***	74	2.2	64	2.1	84	5.45
	Read (60k tps)	54	1.9	45	1.9	68	1.6
	Write (20k tps)	26	0.8	22	0.8	41	1

4.2 Multiple Cassandra Clusters

In the following set of experiments, we consider the case of deploying an increasing number of Cassandra clusters on the test bed, specifically: 1, 2 and 4 clusters.

The left part of Figure 4 shows the CPU utilization for the mix work load for Cassandra clusters deployed with VMware VMs. The lower part of Figure 4 shows the CPU utilization for the mixed work load for Cassandra clusters deployed using Docker containers. As shown in Figure 1, the maximum number of transactions per second for the mix work load is approximately 30% higher for containers compared to VMs. Figure 4 shows that for a certain number of transactions per second the CPU utilization is a bit higher when using VMs compared to containers. The difference in CPU utilization for VMs and containers is, however, not very significant. For both the container and the VM case, the CPU utilization increases when the number of Cassandra clusters increases.

The upper part of Figure 5 shows the CPU utilization for the write work load for Cassandra clusters deployed with VMware VMs. The lower part of Figure 5 shows the CPU utilization for the write work load for Cassandra clusters deployed using Docker containers. As shown in Figure 1, the maximum number of transactions per second for the write work load is 20-25% higher for containers compared VMs. Figure 5 shows that for a certain number of transactions per second the CPU utilization is a bit higher when using VMs compared to containers. The difference in CPU utilization for VMs and containers is, however, not very significant. For both the container and the VM case, the CPU utilization increases when the number of Cassandra clusters increases.

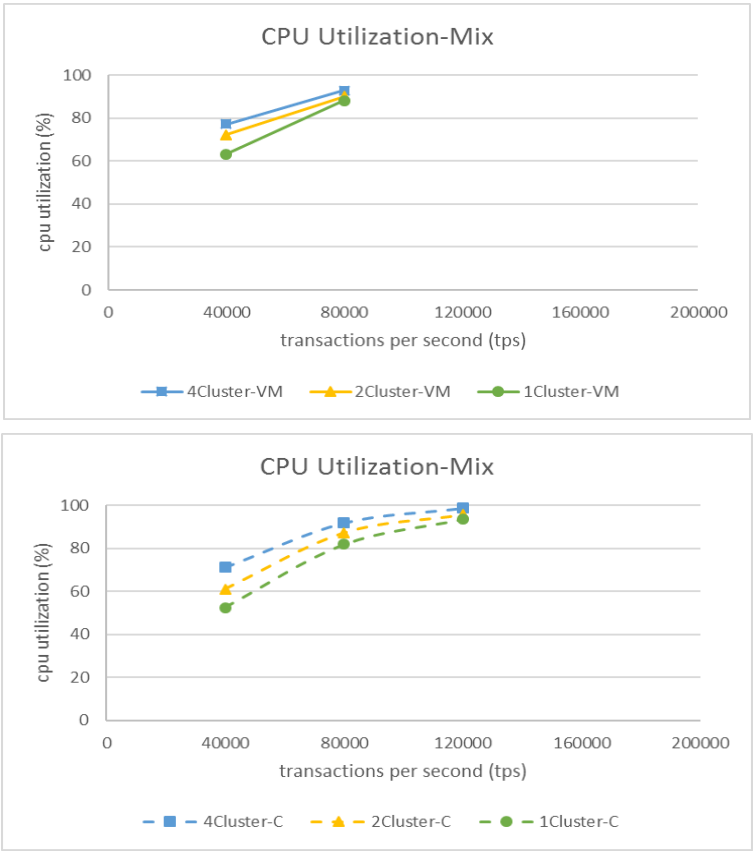

Figure 4: CPU utilization for the mixed work load for VMs (upper) and containers (lower) for different cluster sizes.

The upper part of Figure 6 shows the CPU utilization for the read work load for Cassandra clusters deployed with VMware VMs. The lower part of Figure 6 shows the CPU utilization for the read work load for Cassandra clusters deployed using Docker containers. As shown in Figure 1, the maximum number of transactions per second for the read work load is more than 40% higher for containers compared VMs. Figure 6 shows that for a certain number of transactions per second the CPU utilization is a bit higher when using VMs compared to containers. The difference in CPU utilization for VMs and containers is, however, not very significant. For both the container and the VM case, the CPU utilization increases when the number of Cassandra clusters increases.

The upper part of Figure 7 shows the mean latency for the mix work load for Cassandra clusters deployed with VMware VMs. The lower part of the figure shows the mean latency for the mix work load for Cassandra clusters deployed using Docker containers. Figure 7 shows that for a certain number of transactions per second the latency is higher when using VMs compared to containers. For both the container and the VM case, the mean latency increases when the number of Cassandra clusters increases.

The upper part of Figure 8 shows the mean latency for the write work load for Cassandra clusters deployed with VMware VMs. The lower part of the figure shows the mean latency for the write work load for Cassandra clusters deployed using Docker containers. Figure 8 shows that for a certain number of transactions per second the latency is higher when using VMs compared to containers, particularly when the number of transactions is close to maximum. For both the container and the VM case, the mean latency increases when the number of Cassandra clusters increases.

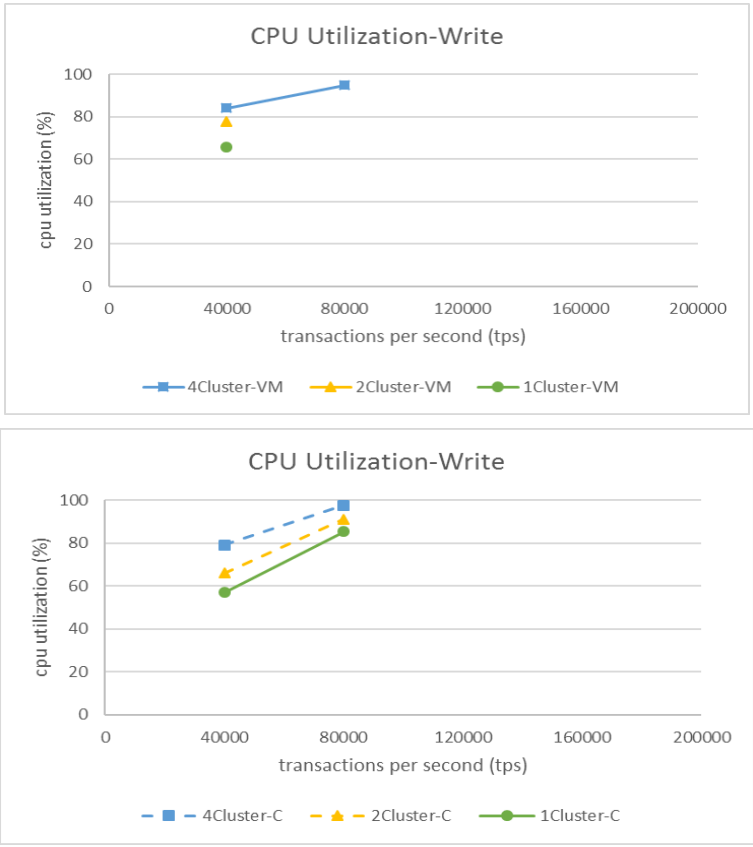

Figure 5: CPU utilization for the write work load for VMs (upper) and containers (lower) for different cluster sizes.

The upper part of Figure 9 shows the mean latency for the read work load for Cassandra clusters deployed with VMware VMs. The lower part of the figure shows the mean latency for the read work load for Cassandra clusters deployed using Docker containers. Figure 9 shows that for a certain number of transactions per second the latency is higher when using VMs compared to containers, particularly when the number of transactions is close to maximum. For both the container and the VM case, the mean latency increases when the number of Cassandra clusters increases.

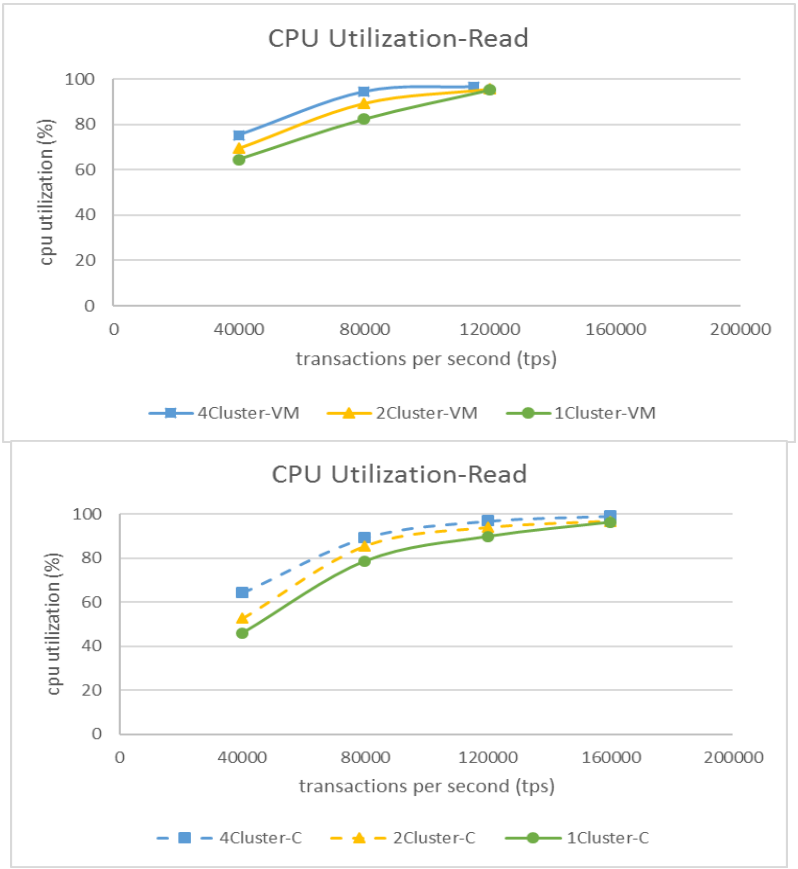

Figure 6: CPU utilization for the read work load for VMs (upper) and containers (lower) for different cluster sizes.

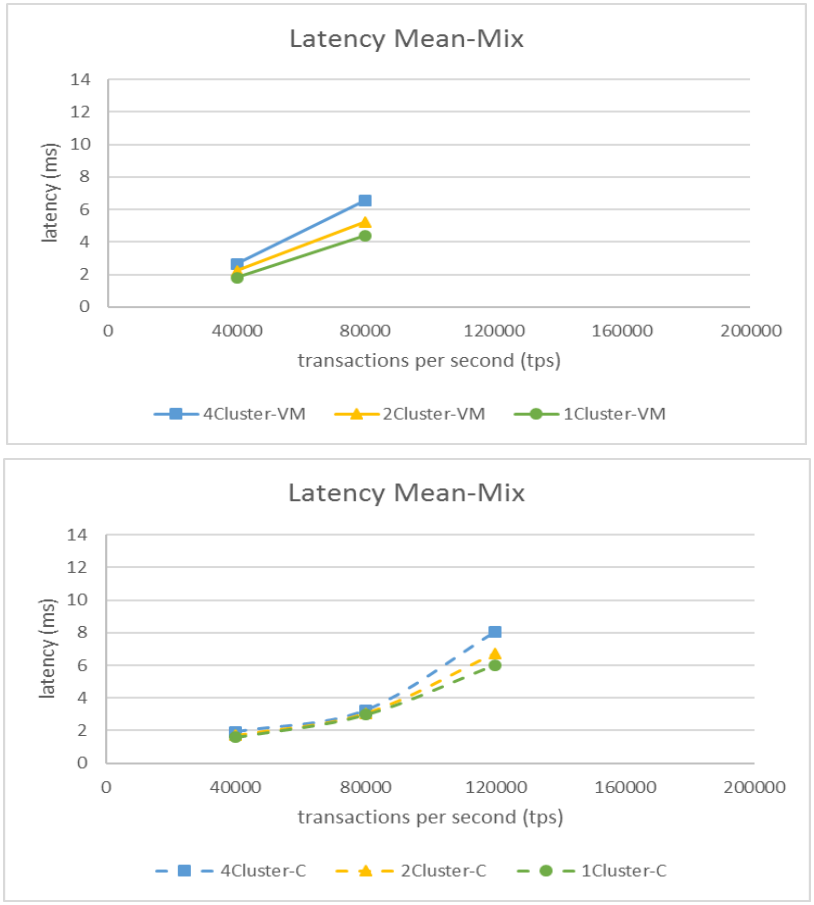

Figure 7: Mean latency for the mixed work load for VMs (upper) and containers (lower) for different cluster sizes.

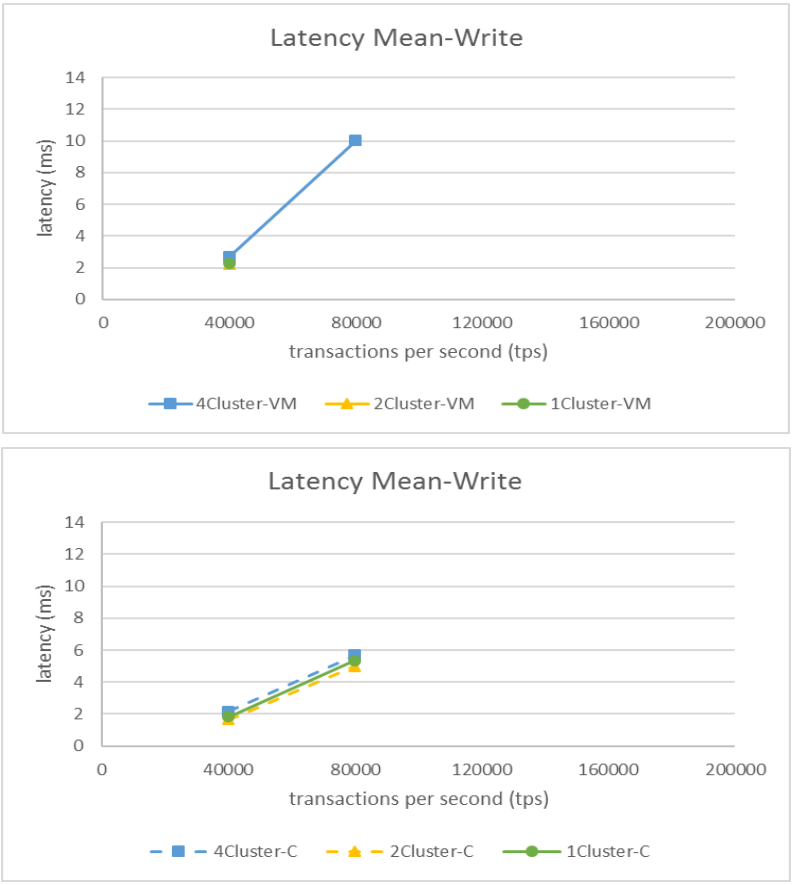

Figure 8: Mean latency for the write work load for VMs (upper) and containers (lower) for different cluster sizes.

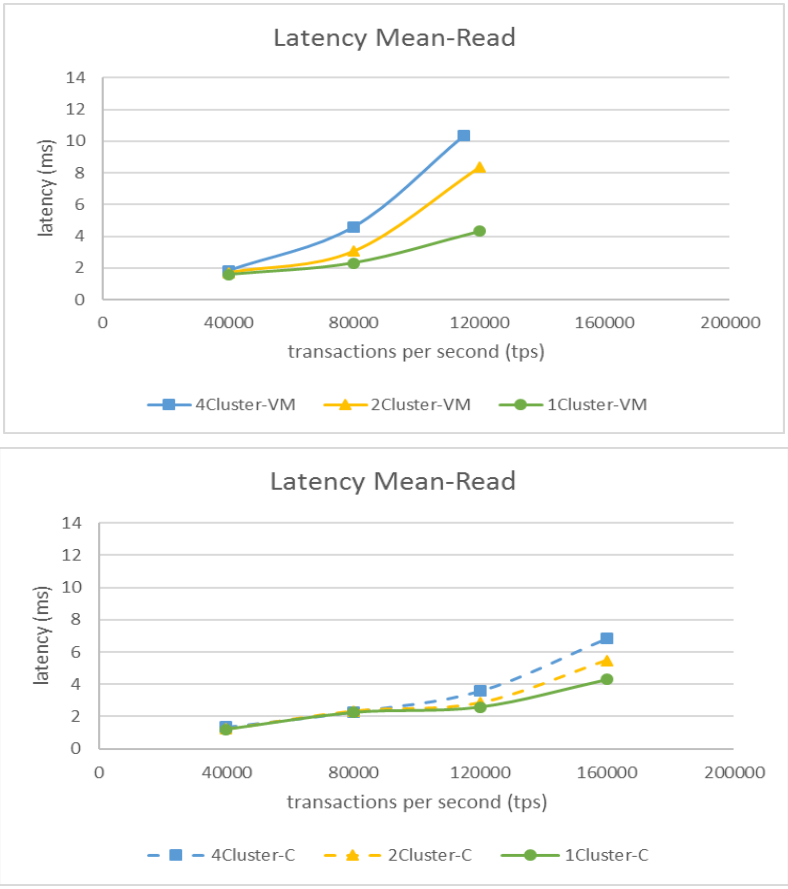

Figure 9: Mean latency for the read work load for VMs (upper) and containers (lower) for different cluster sizes.

5. Conclusions and Future Work

In this paper, we investigate which solution is better for a distributed Cassandra database: non-virtualized, virtualized (VMware) or Docker? The overall result showed that the biggest issue with running the virtualized VMware solution, is the significant resource and operational overheads which affects the performance of the application. The Docker solution seems to address the challenges of virtualization by packaging the applications and their dependencies into lightweight containers. According to the result, the Docker solution consumed fewer resources and operational overheads compared to the virtualized VMware solution. Also, the performance in terms of maximum number of transactions per second was at least as high for the Docker solution as for the non-virtualized case.

We also investigated the effect of dividing a Cassandra database into multiple independent clusters. It turned out that write operations benefit from having multiple Cassandra clusters, whereas the read performance is highest for one cluster. The reason for this is that there are two aspects that affect the performance in the case of multiple clusters: first the overhead increases when the number of clusters increases for both the container and the VM case. Second, the VMware hypervisor and the Docker system use affinity scheduling in the sense that a VM or a Docker container tends to be scheduled on a certain subset of the processor cores, which increases the cache hit ratio in case of multiple Cassandra clusters. For the read work load, affinity scheduling has no, or little, effect on the cache it ratio and performance. However, for the write work load, affinity scheduling increases the cache hit ratio and performance.

Even though the Docker container solution is showing very low overhead and system resource consumption, it suffers from security problems when storing data, which is crucial for database protection. Comparing containers with virtual machines, containers cannot be secure candidates for databases because all

containers share the same kernel and are therefore less isolated than VMs. A bug in the kernel affects every container and results in significant data loss. On the other hand, hypervisor-based virtualization is a mature and (relatively) secure technology.

According to our results, hypervisor-based virtualization suffers from noticeable overhead which effects the performance of the databases. A first recommendation, valid for deployments in a single tenant environments, is to use containers directly on the host, without any other virtualization level in the middle. Orchestration frameworks like Kubernetes could be used to implement automation, high availability and elasticity (like in any other classical cloud setting). As a second advice, since both containers and virtual machines have their set of benefits and drawbacks, an alternative solution could be to combine the two technologies. That would increase isolation, important in multitenancy environments, and allow us to benefit from the full in-memory features of containers. In the future, we plan to investigate such alternative solutions by running containers inside virtual machines running Cassandra workload. In this way, we may get the benefits of both the security of the virtual machine and the execution speed of containers.

6. REFERENCES

[1] L. Yu, L. Chen, Z. Cai, H. Shen, Y. Liang, Y. Pan, "Stochastic Load Balancing for Virtual Resource Management in Datacenters," Published in IEEE Transactions on Cloud Computing (Volume: PP, Issue: 99), 2016, pp 1-14.

[2] T. Adufu, J. Choi, Y. Kim, "Is Container-Based Technology a Winner for High Performance Scientific Applications?" Published in APNOMS, 2015, pp 507-510.

[3] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Bavier, L. Peterson, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," Published in EuroSys conference, 2007, pp. 275-287.

[4] J. Li, et al. "Performance Overhead among Three Hypervisors: An Experimental Study using Hadoop Benchmarks," Published in IEEE international congress on Big Data, 2013, pp 9-16.

[5] S. G. Soriga and M. Barbulescu, "A Comparison of Performance and Scalability of Xen and KVM Hypervisors," Published in RoEduNet international conference 12th edition, 2013, pp 1-6.

[6] J. Hwang, S. Zeng, F. Wu, T. Wood, "A Component-Based Performance Comparison of Four Hypervisors," Published in IFIP/IM, 2013, pp 269-276.

[7] J. Che, Q. He, Q. Gao, D. Huang, "Performance Measuring and Comparing of Virtual Machine Monitors," Published in EUC international conference, 2008, pp 381-386.

[8] S. Shirinbab, L. Lundberg, D. Ilie, "Performance Comparison of KVM, VMware and XenServer using a Large Telecommunication Application," Published in the 5th international conference on Cloud Computing, GRIDs, and Virtualization, 2014, pp 114-122.

[9] R. Dua, A. R. Raja, D. Kakadia, "Virtualization vs Containerization to support PaaS Cloud Engineering," Published in IC2E conference, 2014, pp. 610- 614.

[10] LXC, "Linux Containers," 2016, https://linuxcontainers.org/

[11] C. Pahl, "Containerization and the PaaS Cloud," Published in IEEE Cloud Computing (Volume: 2, Issue: 3), 2015, pp 24-31.

[12] R. Rosen, "Linux Containers and The Future Cloud," Linux J, 2014.

[13] R. Wu, A. Deng, Y. Chen, E. Blasch, B. Liu, "Cloud Technology Applications for Area Surveillance," Published in NAECON conference, 2015, pp 89-94.

[14] OpenVZ, "OpenVZ Virtuozzo Containers," 2016, https://openvz.org/

[15] K. Kolyshkin, "Virtualization in Linux," Whitepaper, OpenVZ, 2006.

[16] B. I. Ismail et al. “Evaluation of Docker as Edge Computing Platform,” Published in ICOS conference, 2015, pp 130-135.

[17] Docker, “Docker,” 2016, https://www.docker.com/

[18] C. Anderson, ”Docker: Software Engineering”, Software IEEE, (Volume:32, Issue: 3), 2015.

[19] C. Kan, “DoCloud: An Elastic Cloud Platform for Web Applications Based on Docker,” Published in ICACT, 2016, pp 478-483.

[20] E. N. Preeth, J. Mulerickal, B. Paul, Y. Sastri, “Evaluation of Docker Containers Based on Hardware Utilization,” Published in ICCC conference, 2015, pp 697-700.

[21] A. Gkortzis, S. Rizou, D. Spinellis, “An Empirical Analysis of Vulnerabilities in Virtualization Technologies,” Published in Cloud Computing Technology and Science conference, 2016, pp. 533-538.

[22] C. Mohmet and M. Buyukkececi, “Big Data Challenges in Information Engineering Curriculum," Published in EAEEIE conference, 2014, pp 1-4.

[23] R. Jain, S. Iyengar, A. Arora, “Overview of Popular Graph Databases,” Published in: ICCCNT conference, 2013, pp 1-6.

[24] Cassandra, “Cassandra,” 2016, http://cassandra.apache.org/

[25] E. Hewitt, Cassandra: The Definitive Guide. O’Reilly Media, 2010.

[26] J. R. Lourenco, et al. “NoSQL in Practice: A Write-Heavy Enterprise Application,” Published in IEEE international congress on Big Data, 2015, pp 584-591.

[27] A. Chebotko, A. Kashlev, S. Lu, ”A Big Data Modeling Methodology for Apache Cassandra,” Published in IEEE international congress on Big Data, 2015, pp 238-245.

[28] R. P. Goldberg, “Survey of Virtual Machines Research,” Computer, 1974.

[29] D. Liu, L. Zhao, “The Research and Implementation of Cloud Computing Platform Based on Docker,” Published in ICCWAMTIP conference, 2014, pp 475-478.

[30] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” Published in IEEE Cloud Computing (Volume: 1, Issue: 3), 2015, pp 81-84.

[31] Nitin Naik, “Migrating from Virtualization to Dockerization in the Cloud Simulation and Evaluation of distributed Systems,” Published in a conference of the Maintenance and Evolution of Service-Oriented and Cloud-based Environments, 2016, pp. 1-8.

[32] M. Raho, A. Spyridakis, M. Paolino, D. Raho, “KVM, Xen and Docker: A Performance Analysis for ARM based NFV and Cloud Computing,” Published in AIEEE conference, 2015, pp 1-8.

[33] J. Zhang, X. Lu, D.K. Panda, “Performance Characterization of Hypervisor -and Container-Based Virtualization for HPC or SR-IOV Enabled InfiniBand Clusters,” Published in Parallel and Distributed Processing Symposium Workshops, 2016, pp. 1777-1784.

[34] M. T. Chung, N. Q. Hung, M. T. N. Thoai, “Using Docker in high Performance Computing Applications,” Published in Communications and Electronics conference, 2016, pp. 52-57.

[35] J. P. Walters, et al. “GPU Passthrough Performance: A Comparison of KVM, Xen, VMware ESXi, and LXC for CUDA and OpenCL Applications,” Published in the 7th international conference on Cloud Computing, 2014, pp 636-643.

[36] N. Leavit, “Will NoSQL Database Live Up to Their Promise?” Published in IEEE Computer, 2010, pp 12-14.

[37] J. Sipke, B. waaij, R. Meijer, “Sensor Data Storage Performance: SQL or NoSQL, Physical or Virtual,” Published in CLOUD, 2012, pp 431-438.

[38] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, “An Updated Performance Comparison of virtual Machines and Linux Containers,” Published in ISPASS conference, 2015, pp 171-172.

[39] G. Martins, P. Bezerra, R. Gomes, F. Albuquerque, “Evaluating Performance Degradation in NoSQL Databases Generated by Virtualization,” Published in LANOMS, 2015, pp 84-91.

[40] E. Dede, B. Sendir, P. Kuzlu, J. Hartog, M. Govindaraju, ”An Evaluation of Cassandra for Hadoop,” Published in CLOUD, 2013, pp. 494-501.

[41] J. Heo, R. Taheri, “Virtualization Latency-Sensitive Applications: Where Does the Overhead Come From?, ” Published in VMware Technical Journal, 2013, https://labs.vmware.com/vmtj/virtualizing-latency-sensitive-applications-where-does-the-overhead-come-from

[42] R. Morabito, J. Kjällman, and M. Komu, ”Hypervisors vs. Lightweight Virtualization: A Performance Comparison” In Proceedings of the 2015 IEEE International Conference on Cloud Engineering (IC2E '15). IEEE Computer Society, Washington, DC, USA, 386-393.

[43] E. Casalicchio, V. Perciballi “Measuring Docker Performance: What a Mess!!!” In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (ICPE '17 Companion). ACM, New York, NY, USA, 11-16.

[44] E. Casalicchio and V. Perciballi, "Auto-Scaling of Containers: The Impact of Relative and Absolute Metrics," 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, AZ, 2017, pp. 207-214.

[45] Zhanibek Kozhirbayev, Richard O. Sinnott, A performance comparison of container-based technologies for the Cloud, Future Generation Computer Systems, v. 68, 2017, Pages 175-182, ISSN 0167-739X,

[46] S. McDaniel, S. Herbein and M. Taufer, "A Two-Tiered Approach to I/O Quality of Service in Docker Containers," 2015 IEEE International Conference on Cluster Computing, Chicago, IL, 2015, pp. 490-491.

[47] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), San Francisco, CA, 2016, pp. 999-1000.

[48] K. Ye and Y. Ji, "Performance Tuning and Modeling for Big Data Applications in Docker Containers," 2017 International Conference on Networking, Architecture, and Storage (NAS), Shenzhen, 2017, pp. 1-6.

[49] V. Tarasov et al., "In Search of the Ideal Storage Configuration for Docker Containers," 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W), Tucson, AZ, 2017, pp. 199-206.

[50] R. Dua, V. Kohli, S. Patil and S. Patil, "Performance analysis of Union and CoW File Systems with Docker," 2016 International Conference on Computing, Analytics and Security Trends (CAST), Pune, 2016, pp. 550-555.

[51] E. W. Biederman. Multiple instances of the global Linux namespaces. In 2006 Ottawa Linux Symposium, 2006.

[52] Datastax, New tocken allocation algorithm in Cassandra 3.0, last check 2018/02/20 https://www.datastax.com/dev/blog/token-allocation-algorithm