Indonesian Journal of Electrical Engineering and Computer Science
Vol. 7, No. 1, July 2017, pp. 205 ~ 213
DOI: 10.11591/ijeecs.v7.i1.pp205-213

An Empirical Performance Evaluation of Docker Container, Openstack Virtual Machine and Bare Metal Server

Jyoti Shetty*[1], Sahana Upadhaya[2], Rajarajeshwari HS[3], Shobha G[4], Jayant Chandra[5]
Department of Computer Science and Engineering, R V College of Engineering, Bangalore
Corresponding author, e-mail: sjyothi.12@gmail.com*[1], sahana496@gmail.com[2]

Abstract

Server virtualization is a fundamental technological innovation that is used extensively in IT enterprises. Server virtualization enables creation of multiple virtual machines on single underlying physical machine. It is realized either in form of hypervisors or containers. Hypervisor is an extra layer of abstraction between the hardware and virtual machines that emulates underlying hardware. In contrast, the more recent container-based virtualization technology runs on host kernel without additional layer of abstraction. Thus container technology is expected to provide near native performance compared to hypervisor based technology. We have conducted a series of experiments to measure and compare the performance of workloads over hypervisor based virtual machines, Docker containers and native bare metal machine. We use a standard benchmark workload suite that stresses CPU, memory, disk IO and system. The results obtained show that Docker containers provide better or similar performance compared to traditional hypervisor based virtual machines in almost all the tests. However as expected the native system still provides the best performance as compared to either containers or hypervisors.

Keywords*: Docker Container, Performance Evaluation, Openstack Virtual Machine, phoronix test suit.*

1. Introduction

Linux Containers (LXC), are the building blocks that formed the foundation for containerization technology. LXC combined the use of kernel cgroups and namespaces to implement lightweight process isolation. Cgroups allows for isolating and tracking resource utilization, and namespaces allows groups to be separated so they cannot see each other [1]. Docker is an opensource container technology that has made it easy for developers and system administrators to create and manage containers. It initially used LXC as its default driver. It later developed libcontainer for this purpose. In Docker the applications can be broken into functional components, each running separately in a container with all its dependencies. These containers can then be run on any architecture. These architectures should however have Docker installed on them. Scaling and updating the components now becomes fairly simple. While running applications in Docker containers, there is no need to worry about setting it up and maintaining different environments or different tools for each language. We can instead focus on fixing issues, developing good code, adding new features and shipping the software. Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run i.e. code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that it will always run the same, regardless of the environment it is running in [2]. Docker makes it easy to share and collaborate on applications. Storing, distributing and managing Docker images is done with the help of Docker hub. Docker hub is an online repository for all Docker images [2].

In this work, we carry out a series of experiments to measure and compare the performance of applications over bare metal, hypervisor based virtual machine and Docker container. These tests help us to understand the performance implications of the two major types of virtualization technologies - containers and hypervisors.

Received March 10, 2017; Revised May 20, 2017; Accepted June 6, 2017

2. Research Method

In this section, first an overview of Docker technology is presented with aim to understand the features of the upcoming popular container technology. We also present a detailed comparison between traditional hypervisor based virtualization and Docker containers.

2.1. Docker Architecture

Docker is based on the concept of each container running in its own protected isolated space. This isolation allows many containers to be run at the same time. Containers do not need a hypervisor layer. As a result of this they are extremely lightweight and provide a way for maximum utilization of the underlying hardware. Docker components include

2.2. Docker Client and Daemon

Docker is based on client-server architecture. Docker daemon or server is responsible for all the actions related to containers like building, running and distributing them. The daemon receives commands from the Docker client through cli or REST API's. The architecture of the Docker client and Daemon is shown in Figure 1.

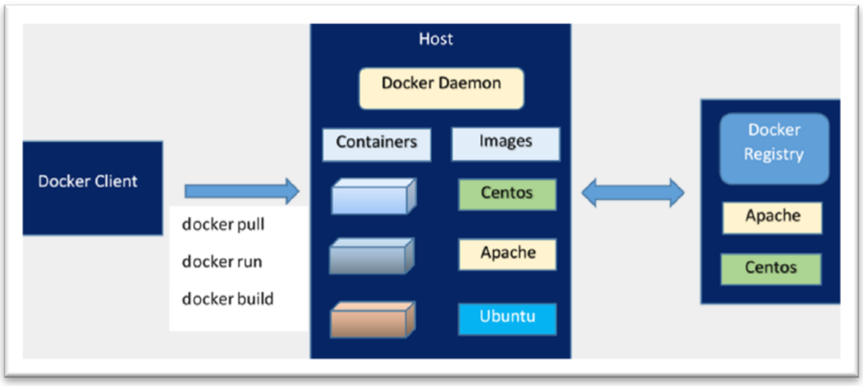


Figure1. Docker architecture [3].

2.3. Images

Images are the basic building blocks of Docker. Every container is launched from images. Images can be configured with applications and used as a read-only template for creating containers. For example, an image could contain an Ubuntu operating system with Apache and a web application installed in it. The images are organized in the form of layers. Every change in an image is added as a layer on top of it. Docker makes use of union file systems to combine these layers into a single image. Union file systems allow files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.

One of the main reasons why Docker is so lightweight is because of these layers. If the need arises to update the current Docker image, for example, installing a newer version of java jdk, then a new layer gets built. Instead of replacing or entirely building the complete image, only that layer gets added or updated. When distributing the image, only the updated version can be shared, making the distribution of images faster and simpler [3].

2.4. Docker Registries

Docker Registry is a repository for Docker images. Building and sharing images becomes extremely simple. A registry can be made public or private. This registry is called Docker hub. There is a huge collection of existing images which can be downloaded with a simple command:

Docker pull <image name>

Similarly, to upload an image into Docker hub, the following command can be used.

Docker push <image name>

Searching and downloading of images can be done with ease using the Docker client [4].

2.5. Containers

Docker containers are similar to a directory. They provide the execution environment needed for an application to run. Containers are created from images. They are the writable layer of the image. Containers consist of an operating system, user-added files and meta-data. Applications can be packaged in a container. They can also be committed and made as an image that can be used by other containers. Containers can be run, started, stopped, moved and deleted with simple commands [3].

3. Comparison between Docker Container and Virtual Machine

Though Docker is sometimes refered to as light-weight VMs, they are not VMs. The underlying technology as discussed in the above section is what differentiates both of them. The major differences between the two are explained.

3.1. Virtualization

On the VM side, hypervisors make portions of the hardware available. There are two types of hypervisors: Type 1 runs as an application on a host operating system and Type 2 directly runs on the bare metal of the hardware. Xen, VMware ESX are examples of Type 1 whereas Oracle's Virtual Box and VMware servers are of Type 2 [5]. The two types of hypervisor based virtualization are shown in Figure 2(a) and Figure 2(b).

On the other hand, containers make available only protected portions of the operating system. If two containers are running at the same time, then they both do not know that they are sharing resources because each one has its own abstracted operating system layer, process, network layer, etc. Docker virtualization is shown in Figure 3.

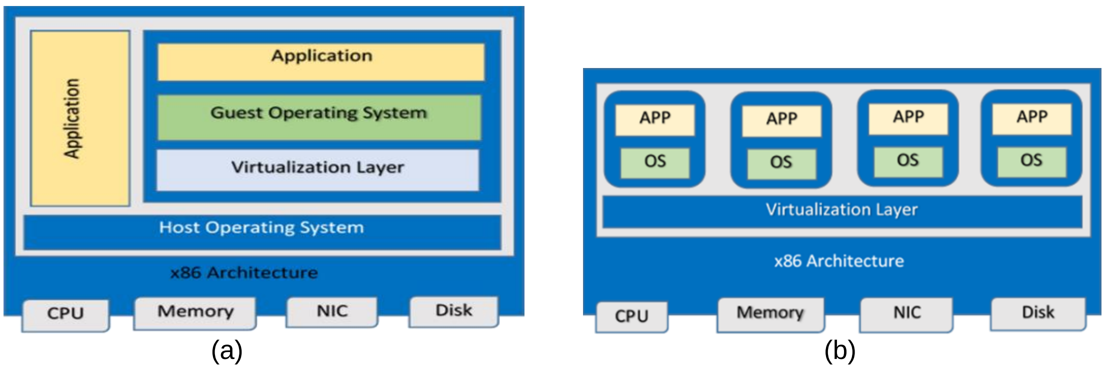


Figure 1. (a) Type 1 virtualization, (b) Type 2 virtualization [5]

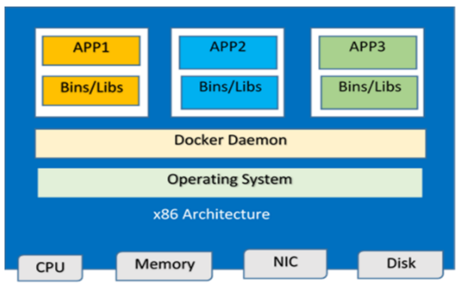


Figure 3. Docker operating system-level virtualization [6]

3.2. Operating Systems and Resources

Hypervisor-based virtualization provides access to hardware only. So we would have to install our own fully-fledged OS in it. If there are multiple VMs running, then it would soon exhaust the server resources like CPU, RAM and bandwidth. Another factor to be considered is the overhead of booting the entire OS.

Containers, on the other hand share the host kernel. As a result of this, it would just contain a minimal image of the operating system which primarily includes the filesystem and binary files. They mainly run in spaces isolated from each other and from certain parts of the host OS. They have two significant benefits over VMs. First, resource utilization is much more efficient. So if a container in not running anything, then it is not consuming the system resources. Second, there is no overhead of booting OS. They are simple to create and destroy. While it takes the VM about the same time to boot as a normal system which is around 20s, it takes only 500ms to get a container up and running.

3.3. Isolation for Performance and Security

Processes executing in a Docker container are isolated from processes running on the host OS or in other Docker containers. Nevertheless, they all are running on the same kernel. Docker leverages LXC to provide separate namespaces and also uses Control Groups which are already present in the Linux kernel. Docker daemon itself poses a potential threat as it runs on the host OS with root privileges [7].

Although the type of isolation provided is strong overall, it is definitely not as strong as that enforced by virtual machines at the hypervisor level. If the host kernel itself goes down, then so do all the containers. VMs have another advantage over containers in that they are widely used in production environments and have more maturity compared to containers which are still in their infancy stage. Docker and its supporting technologies have not seen nearly as much action as VMs. Table 1 shows the overall comparison between Virtual Machines and Docker virtualization.

Table 1. Comparison between VM and Docker Container

VM	DOCKER CONTAINER
It is based on Hardware level virtualization.	It is based on OS level virtualization.
It has a full-fledged OS running and therefore has its own kernel support.	It has only the minimal OS image and shares the host kernel.
More secure since any attack would have to pass the VM kernel, hypervisor layer and then the host kernel.	Reduced security in containers since the container and the host share the same kernel.
Better isolation between individual processes.	Isolation between processes is lower than VM.
It takes several minutes to create and launch a VM.	It takes only a few seconds to create and launch a container.
It consumes a lot of system resources and occupies more disk space.	There is better resource utilization and minimal disk utilization for containers.
VM supports any type of guest OS, viz. Windows, Linux, Mac OS X.	Only different distros of Linux can be run.

4. Methodology

Here, we compare the performance of Docker containers and VM against that of the native non-virtualized system. The native bare metal system is used as a base to evaluate the performance overhead of virtualization. Numerous tests are carried out to measure the overhead of virtualization. These benchmark tests include CPU, system, memory and disk workloads provided by Phoronix test suite [8].

All the tests were performed on an HP server with two Intel Xenon E5-2620 v3 processors at 2.40 GHz at a total of 12 cores and 128 GB RAM. Hyper-threading was disabled. Centos 7 64-bit with Linux kernel 3.10.0 was used to perform all the tests. To maintain consistency and uniformity, the same operating system, Centos 7, was used as the base image for all Docker containers .We created an OpenStack Virtual instance with QEMU-KVM as the System Layer and running Centos 7 cloud image. The VM was configured with 12 vCPUs and sufficient RAM to support the benchmark working.

5. Results and Discussion

An empirical evaluation of the performance of traditional hypervisor based VMs, Linux Docker containers and bare metal is presented in this section.

5.1. CPU Intensive Workloads

5.1.1. Gzip Compression

Compression is the process of encoding information in fewer bits. It a processor intensive task and therefore has been used as one of the workloads. Gzip compression makes use of DEFLATE, a data compression algorithm which is a combination of LZ77 and Huffman coding [9]. This test measures the time needed to compress a file using Gzip compression. The file size used for the compression test is 2GB. Figure 4(a) shows the results of this test. The bare metal system and Docker show comparable performance while VM is approximately 27% slower.

5.1.2. MAFFT Alignment

Multiple alignment using Fast Fourier Transforms [10] (MAFFT) is an important tool for computational analysis of nucleotide or amino acid sequences. MAFFT is among the fastest methods available for multiple alignment. The CPU time taken to perform an alignment of 100 pyruvate decarboxylase sequences is tested here. Again the performance of Docker and bare metal remained almost the same, while VM showed a reduced performance (by approximately 28%). The performance comparison is shown in Figure 4(a).

5.1.3. Himeno Benchmark

The Himeno benchmark [11] is a linear solver of pressure Poisson using a point-Jacobi method developed by Dr Rayutaro Himeno. This benchmark was developed to evaluate the performance of incompressible fluid analysis code. This test is determined by the performance of the computer processor, especially memory bandwidth. It is measured in terms of MFLOPS.
The resulting graph in Figure 4(b) indicates that all three systems, bare metal, Docker and VM do not show much variation in performance. This might be due to the minimal OS involvement during execution. Bare metal outperforms Docker containers and virtual machines by a mere 6%. Docker containers still performs better than virtual machines (although only by ~2%)

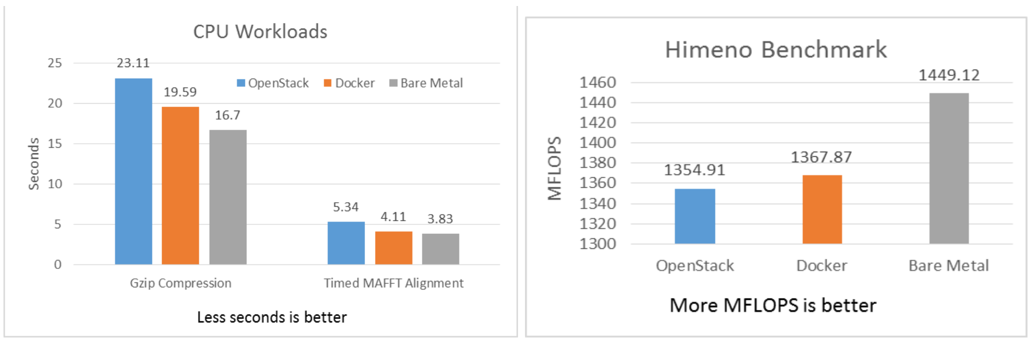


Figure 4. CPU workloads: (a) Gzip compression and Timed MAFFT Alignment performance comparison (b) Himeno Poisson Pressure Solver performance comparison

5.2. Memory Intensive Workloads

5.2.1. RAM Speed

The three different execution environments' memory (RAM) performance was compared with three programs to test Integer addition, Integer scale, and Integer copy. The results yielded comparable performance of all three environments as shown in Figure 6.

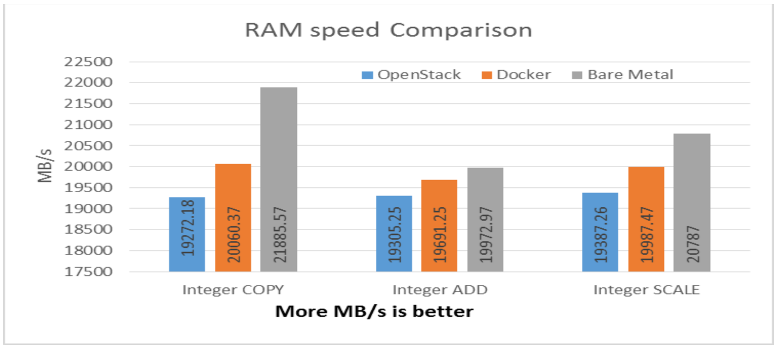


Figure 6. RAM Speed (System Memory Performance) comparison

5.2.2. STREAM benchmark

The STREAM benchmark [12] is a synthetic benchmark program written in Fortran 77 and measures sustainable memory bandwidth and the corresponding computation rate for four different vector kernels, namely, copy, scale, add and triad. The memory bandwidth is measured in MB/s. In our analysis, only stream copy, scale and add have been tested.

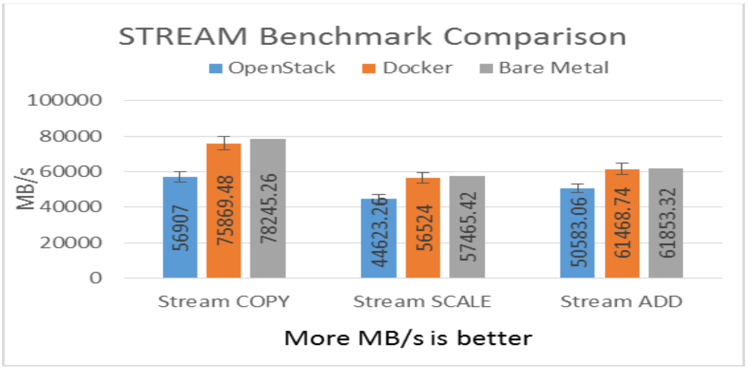


Figure 5. Stream Benchmark Performance Comparison

The STREAM benchmark is specifically designed to work with datasets much larger than the available cache on any given system, so that the results are more indicative of the performance of very large, vector style applications. Figure 5 shows the results of the test. While Docker and bare metal perform almost equal in terms of number of MB/s, KVM is almost 21% slower on an average.

5.3. Disk Intensive Workload

5.3.1. PostMark Benchmark

This is a test of NetApp's PostMark benchmark [13] designed to simulate small-file testing similar to the tasks endured by web and mail servers. This test profile was set for PostMark to perform 25,000 transactions with 500 files simultaneously with the file sizes ranging between 5 and 512 kilobytes.The results of the test are shown in Figure 7. The performance is compared in terms of Transactions Per Second (TPS). The tests again favoured bare metal and VM performed the worst (by approximately 27%).

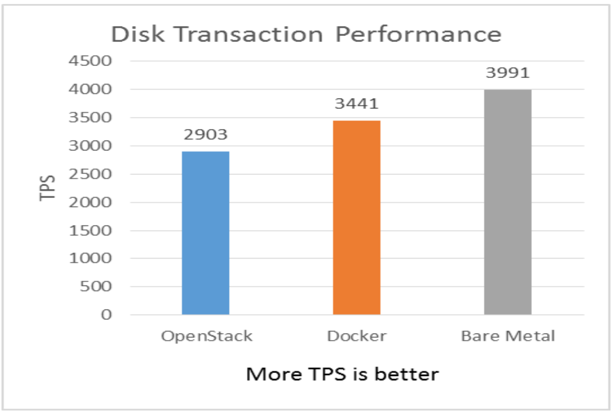
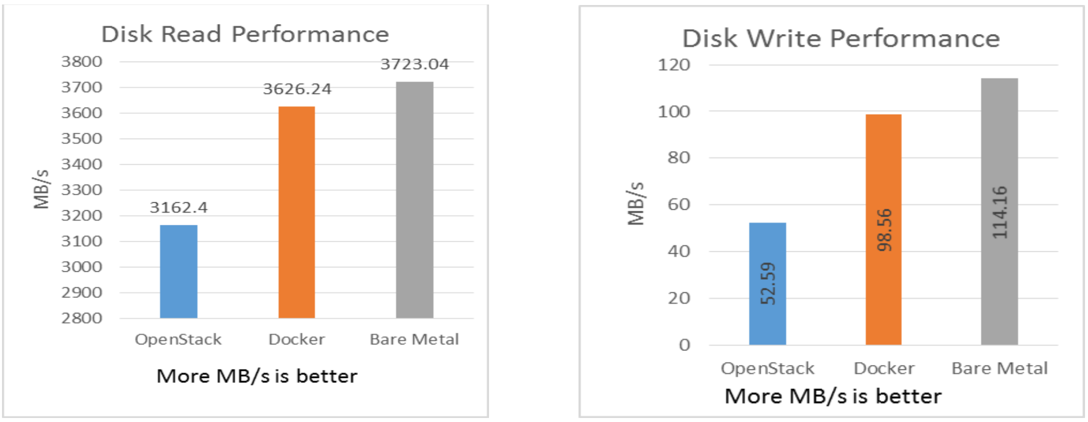


Figure 7. Postmark Benchmark Performance Comparison

5.3.2. Disk Read/Write

The IOzone benchmark [14] is used to test the hard disk drive performance. For testing the Read performance of the system, a record size of 1MB and a file size of 512 MB was used. To test the Write performance of the system, the record and file size are used. We can infer from Figure 8(a) and Figure 8(b) that the performance of bare metal and Docker are much better than the VM. The disk write performance of a virtual machine is reduced by more than half of that of bare metal (approximately 54%) while Docker container lags behind by only 13%.



(a) (b)

Figure 8. (a) Disk Read Performance Comparison (b) Disk Write Performance Comparison

This can be attributed to the fact that for a disk operation to be performed by the virtual machine, it has to pass through the QEMU layer. This overhead probably reduces the VM's read and write performance.

5.4. Apache Benchmark

5.4.1. Apache Benchmark Tool

Apache Benchmark [15] is a tool for benchmarking Apache Hypertext Transfer Protocol server. This test profile measures how many requests per second a given system can sustain when carrying out 1,000,000 requests with 100 requests being carried out concurrently. We can see from Figure 9 that the throughput for VM is much lesser compared to that of Docker and bare metal. This is due to higher network latency in VM than in Docker or bare metal. The performace of Docker container being slightly (approximately 3%) better than bare metal could not be accounted for due to insufficient information.

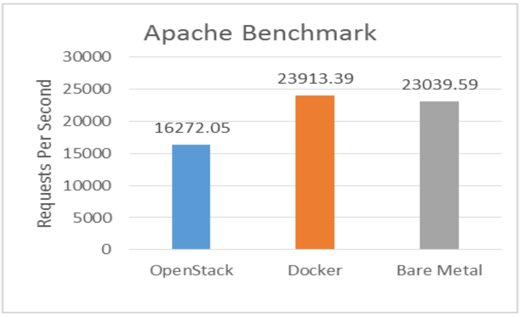


Figure 9. Apache Benchmark comparison

5.5. Overall System Performance

5.5.1. Chess Test Suite

The Chess test suite tests the system’s performance through various AI algorithms for a game of chess. The Figure 10(a) shows the performance test of Crafty [16], an advanced open-source chess engine. Figure 10(b) is a performance test of TSCP, Tom Kerrigan's Simple Chess Program, which is an Artificial Intelligence (AI) Chess Performance benchmark. Both these tests show that Docker container performance almost equals that of bare metal performance while virtual machine shows reduced performance (by approximately 28-30%)

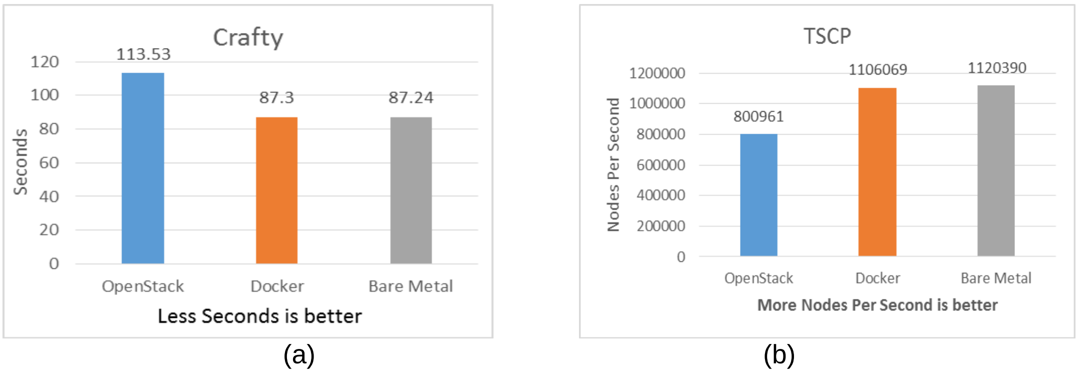


Figure 10. Chess test suite: (a) Crafty (Advanced open-source chess engine) performance comparison (b) TSCP (AI Chess performance) comparison

5.5.2. Sudoku Test Suite

The Sudoku test measures how long it takes to solve 100 Sudoku puzzles. The Sudoku program is written in TCL and determines the system’s computational performance. All three show similar performance as shown in Figure11.

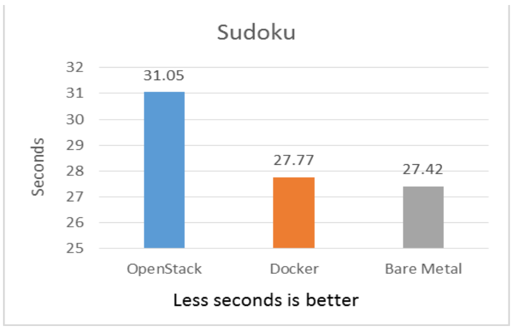


Figure 11. Sudoku program performance comparison

6. Conclusion

Docker is the latest technology in the field of virtualization and it comes with huge benefits for developers and sysadmins. Docker containers are easily scalable and extremely lightweight. Abstraction of the host system away from containerized application is one of the main reasons for their popularity. They also provide simple dependency management and minimal disk utilization. It is also an opensource framework and inexpensive.

A detailed performance evaluation of traditional hypervisor based virtualization, Docker containers and bare metal was carried out in this work. After performing tests on CPU, memory, disk and system, we can see that Docker containers exceed or equal the performance of Openstack KVM virtual instance in every test. While there was not much difference between the performances for CPU and memory workloads, the disk I/O performance of VM was much slower than Docker and bare metal. This was due to the presence of the QEMU layer in the Virtual Machine. The overall system performance of both Docker and VM are comparable.

However, the downside of using containers is their reduced isolation and security level. The reduced security is due to the fact that Docker containers share the host kernel. Virtual machines on the other hand have the hypervisor layer along with the guest OS kernel and therefore have a much superior security.

While considering the choice of virtualization, both of these factors are significant. While Docker can offer better performance compared to VM along with faster boot time, security is not as good as that of virtual machine.

References

[1] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio, IBM Research Report, An Updated Performance Comparison of Virtual Machines and Linux Containers, July 21 2014

[2] Docker Overview, https://www.Docker.com/what-Docker [accessed on 21 June 2017]

[3] Docker Architecture Overview, https://docs.Docker.com/engine/understanding-Docker [accessed on 21 June 2017]

[4] Docker Registry Overview, https://docs.Docker.com/registry [accessed on 21 June 2017]

[5] Virtualization Overview. http://www.vmware.com/pdf/virtualization.pdf. [accessed on 21 June 2017]

[6] Zaheda Haidri, Docker Containers, https://www.datadoghq.com/blog/Docker-performance-datadog/ [accessed on 21 June 2017]

[7] Sahana Upadhya, Jyoti Shetty, Raja Rajeshwari H S, Dr. G Shobha, “A State-of-Art Review of Docker Container Security Issues and Solutions”, American International Journal of Research in Science, Technology, Engineering & Mathematics,17(1), Dec 2016-February 2017, pp. 33-36

[8] Michael Larabel, http://www.phoronix-test-suite.com [accessed on 21 June 2017]

[9] Ziv J., Lempel A., ``A Universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, Vol. 23, No. 3,pp. 337-343.

[10] Himeno Benchmark workload, http://accc.riken.jp/en/supercom/himenobmt/ [accessed on 21 June 2017]

[11] John D. McCalpin, *Advanced Systems Division*, Silicon Graphics, Inc. , http://www.cs.virginia.edu/stream/ref.html [accessed on 21 June 2017]

[12] MAFFT alignment algorithm, http://nar.oxfordjournals.org/content/33/2/511.full [accessed on 21 June 2017]

[13] NettApp PostMark benchmark, http://www.filesystems.org/docs/auto-pilot/Postmark.html [accessed on 21 June 2017]

[14] William Norcott, IOzone Benchmark, http://www.iozone.org/ [accessed on 21 June 2017]

[15] ab - Apache HTTP server benchmarking tool, http://httpd.apache.org/docs/2.2/en/programs/ab.html [accessed on 21 June 2017]

[16] Dr. Robert M. Hyatt, Crafty- open source chess engine, https://www.cis.uab.edu/hyatt/ [accessed on 21 June 2017]