Performance Analysis of Virtual Machines and Containers in Cloud Computing

Rabindra K. Barik
KIIT University,
Bhubaneswar, India
rabindra.mnnit@gmail.com

Rakesh K. Lenka
IIIT-Bhubaneswar,
Bhubaneswar, India
rakeshkumar@iiit-bh.ac.in

K. Rahul Rao
IIIT-Bhubaneswar,
Bhubaneswar, India
b411020@iiit-bh.ac.in

Devam Ghose
IIIT-Bhubaneswar,
Bhubaneswar, India
b411014@iiit-bh.ac.in

Abstract—**Cloud computing provides variety of services with the growth of their offerings. Due to efficient services, it faces numerous challenges. It is based on virtualization, which provides users a plethora computing resources by internet without managing any infrastructure of Virtual Machine (VM). With network virtualization, Virtual Machine Manager (VMM) gives isolation among different VMs. But, sometimes the levels of abstraction involved in virtualization have been reducing the workload performance which is also a concern when implementing virtualization to the Cloud computing domain. In this paper, it has been explored how the vendors in cloud environment are using Containers for hosting their applications and also the performance of VM deployments. It also compares VM and Linux Containers with respect to the quality of service, network performance and security evaluation**.

Keywords—Cloud computing; Containers; OS; virtualization; VMs

I. INTRODUCTION

Cloud computing infrastructures will probably be the future internet architecture, due to the advantages of virtualization and many of its related technologies. Cloud computing allows delivering IT infrastructure from which businesses and users are able to access applications from anywhere in the world on demand [1]. Cloud computing services provide resources by means of virtualization [2] and sometimes also Containerization [4]. Implementing these technologies will certainly provide the ultimate data control to the third parties.

Virtualization is a crucial part in the definition of Cloud computing. Resources that are virtualized in the cloud then can be managed more efficiently. Virtualized Cloud Infrastructure provides the abstraction required to make sure that an application or business service model is independent of the underlying hardware such as servers, storage or networks. Cloud computing depends on virtualization technology that requires resources like a server and breaks it into virtual resources called Virtual Machines (VMs). However, some of the extra levels of abstraction in virtualization have been reducing the workload performance. The basic concept of Containerization is to allow different virtual instances and resource to share a single host Operating System (OS) and its depending libraries, drivers or binaries. This concept reduces the amount of the resources wasted during the computational process as each Container only holds the specific and required application and related binaries or libraries. Containers on the other hand, uses the same host OS repeatedly, instead of installations [11].

II. VIRTUAL MACHINES

A VM replicates a server. In a virtualized server, each virtual "guest" includes a complete OS along with drivers, binaries or libraries, and the same application. Virtualization has used even before Cloud computing was activated, and later it has been settled as an important technique for resource-provisioning, multi-tenancy, and system constitution in IaaS. The core techniques of Cloud computing is solving the problem using the virtualized resources.

Modern VM technologies ([3] [14] [15]) allow a single server which is divided into multiple virtual Containers. Each container delivers a secure, isolated and powerful execution environment for the various applications. Apart from giving access to the resources, these environments can be tweaked to contain the whole software and hardware and their integration [22].

Lifecycle management and resource allocation, is done with the help of the available interfaces of the virtualization platforms [11,20].Thus the virtual machines can be programmed as different processes in the operating system and the Quality of Service(QoS) aware scheduler is used to make the virtual machines to share their space time resources. QoS ensures that the applications running inside of the VMs resource reservation.

The Virtual Resource Manager [12] is responsible for the provisioning of a central management for the resources that are apparently virtualized and spread across the physical hosts. It makes available an interface to serve resource requests to resource clients [21]. All these allocation is not necessarily known by the clients.

A *Virtual Machine Scheduler* [12] primarily manages the deployed virtual machines. The VM scheduler runs concurrently with virtual machine monitor and controls the configurations and life cycles of the VMs. Such an interface like the VMware Infrastructure is used to manage the provisioning of the resource, scheduling of those in an encapsulated manner.

Virtual Machine Monitor. Virtual Machine Monitor [13] is defined as the software for multiple VMs which can draw the identical abstraction of the physical machine [16]. At present, there are several popular Virtual Machine Monitors involve Hyper-V, KVM, OpenVZ, VMware and Xen etc. In hardware virtualization, it uses a software - called as "Virtual Machine Manager" or we can say a hypervisor to create a VM that behaves like a real system though [3]. It can then put a whole OS inside the VM. A VM is called as a guest that stays inside the host OS, where the underlying system that enables VMs is the host. In many cases, multiple VMs execute on a single machine, and many hypervisors support migration of a VM to a different machine when requirement of resources is more.

In hardware virtualization, a software is used called as a hypervisor [19] for creation of a VM that acts like a computer system. It can then install an OS on top of each VM. These VMs will run above the hypervisor, which intrinsically runs on a host OS of the computer and as a result it requires the hardware.

Fig. 1 shows a typical architecture of a VM. The fundamental structure of VM is based on the server of the host system's machine. On top of the server, the host OS is present which in turn runs the system hypervisor. The hypervisor is the crucial part and runs the VM which is actually composed of the Guest OS, on top of that, its dependencies and binaries which finally runs an application. Guest OS is converted or translated to Host OS through many techniques like Binary Translation or Hardware Virtualization. The hypervisor runs on bare EsXi or even it can be a part of the OS itself [6, 18].

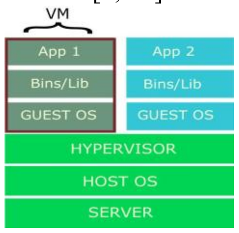

Fig. 1. A VM structure [31]

There are many ways to implement a host VM:

a. On top of the host OS, just as an application by itself. E.g. Oracle Virtual Box.
b. As part of kernel of OS [8]. Examples include Linux operating system.
c. Kernel-based Virtual Machine (KVM).Under the OS itself, as if it were a tiny OS by itself. Examples include Xen and VMware vSphere (also called "ESXi")[17]

These are the following overview of virtualization:

Para-virtualization. The para virtualization machine depends on the drivers and kernels that are virtualized. The kernel of the guest runs on a PVM host will access the system software and hardware through special system calls and then the hypervisor decides what to do with those.[18].

Hardware Virtual Machine. It [15] falls in the lowest rung in the virtualization hierarchy. It makes a system to be completely virtualized minus the requirements of operation drivers on the guest system. The hardware gets interaction calls from the guest without knowing that VM is running in the guest. It tracks the calls made from the guest to the host i.e the system calls like sending data on network interface card. The call is then sent to the hypervisor by the system hardware which then decides what to do with it.

Container Virtualization. It is also termed as virtualization at an operating system level [27]. It makes different containers so as to run different applications and programs so that for security purpose no two applications will interfere in the other's running. All the programs in the containers essentially runs on the same OS. It allows the sharing of the kernel specially between other containers so that all of the system resources were shared without any shortage with minimal overheads [8]. Importantly both Hardware Virtualization and Para-virtualization make use of a hypervisor to connect to the hardware unlike containerization. This makes the use of hypervisors more preferable barring the overhead [15].

III. CONTAINERS

The fundamental concept of Containerization is to make virtual instances, share a single host OS and relevant libraries, drivers or binaries [7, 30]. Fig. 2 shows a typical architecture of a Container system. In this system, instead of the system hypervisor, the Container's own hypervisor engine, i.e the Docker engine sits on top of the host OS. This Container engine is very flexible and light. The Container engine then supports the whole Container image which has its dependencies and binaries to run an application.

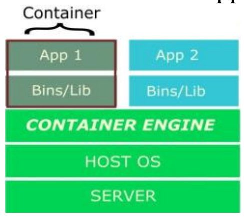

Fig. 2. A Container structure [31]

These instances are often called Containers, and their formation can be described as an isolated separate system (via the file system). As these Containers are based on the OS kernel and also share the kernel, and thus shares files among them, which makes them advantageous over full hardware virtualization.

Linux Containers provide the lightweight virtualization technique by isolating the views of the application in the operating resources. However, Docker, an open source project makes the Container an alternative to the hypervisors. In Docker, Virtual Ethernet has been used to forward the packets among the host and container. Docker also creates an interface virtually named as docker0 on the host. When a Container is created, a pair of peer interfaces connects each of the Containers to the "docker0" bridge. The host and Containers forms a subnet for further configuration is required which allow the Container processes to communicate and expose to the wider world [23]. Linux Containers have been built on the kernel namespaces. This Container is accessed by the clone() system call and it allows creating a separate instance of previous global namespaces. Linux implements the host name namespaces, file system, IPC, PID, network and user [25]. For example, each file system namespace has its own mount table, root directory, same as chroot() but they are powerful. An isolated Container can be created by Namespaces that has no access to objects outside the Container. The various processes that runs out of a Container but apparently appears to run out of a whole some Linux computer but these are actually sharing the host system's kernel with processes originating from different namespaces.

A Container can run a single process where as a VM actually runs a full OS. A Container that runs an application is called an application Container and the one running an OS runs various commands such as sshd, syslogd, cron, init, inetd is a system Container [26][11].

Application Containers usually consume less amount of RAM unlike system Containers. Communication between Containers and the host and vice-versa takes place through the Linux control groups (cgroups) which is used to group processes and control the overall resource consumption. Cgroups are used to minimize the CPU and memory consumption of the Containers. By changing the limits of the cgroups, it can alter and resize the Containers. Cgroups have an ultimate control of processes in the Container. Various management tools for Linux Containers are lmctfy, Warden , LXC, systemd-nspawn, and Docker. Among the tools, Docker has grown very popular. The uniqueness of Docker is that it uses layered file system images run by AUFS (Another Union File System) which uses layered stack of file system and these file systems can be used again and again thus increases efficiency [28,29,24].

Containers can have their basis on a single OS image and thus each Container can have its own set of modified files and dependencies thus resulting in Container images consuming less memory space and so can be conveniently interacted with the cloud. As a result resources do not go to waste as each and every Container contains only its application and its dependent binaries, etc. Containers make use of the same host OS recursively, instead of installing an OS for each guest VM. This is usually referred to as OS-level virtualization. Thus the role of a hypervisor in the OS is instead handled by a Containerization engine, for example, like Docker, which gets installed on top of the host OS [5]. The following are various kinds of mechanism which is associated with Containers.

Chroot. This is the most popular working mechanism with respect to Containers, which creates restricted views in the file system. Chroot (changeroot) is a Linux command which is used to change the root directory to a new directory of the current process and its children [4,24]. Some of the Containers use chroot to isolate and share the file system cross virtual environments. The primary function of the chroot system call is to change the directory that is referred to as/for which it is possible to create separate views of the file system for a process and its child.

Every process includes a pointer which is considered as the root of the file system, and all subsequent processes created by that process inherits this value. The mechanism of chroot is to create isolated instance, which will not see the files created outside its region of chroot tools, like the pbuilder and mock, which are built on top of chroot.

Cgroups. Cgroups is a part of the kernel subsystem. It provides a fine grained control over the sharing of resources like memory, CPU, etc. Most of the Containers use cgroups as their underlying mechanism for resource management. Cgroups also set the limits of these resources based on the usage [4].

Kernel Namespaces. It creates a barrier between Containers at various levels. The net namespace allows each Container to have its network artifacts like IP tables, routing table and loopback interface. The pid namespace allows each Container its own process id [4] [10].

Kernel Virtual Machine (KVM). KVM is a feature of Linux which allows Linux as a type 1 hypervisor running an unmodified guest operating system inside the Linux process. It uses hardware virtualization features in the recent processors to reduce the complexity and the overhead [6]. A VM impersonates a complete server. In a virtualized server environment, each VM guest includes a complete OS along with any binaries, libraries and drivers, and then the actual application on top of the Container.

IV. RELATED WORK

There are various related work that was investigated in terms of performance analysis for VMs and Containers. It was built atop Cloud environment first on Docker which is based on Linux Container and Hypervisor which is VM, It was each analyzed with respect to the size, boot speed, and CPU performance. With this analysis result, Users will be able to understand the characteristics of each platforms, and they will be able to reasonably choose the platforms based on what they need. It was only tested in boot speed and CPU performance [11].

It also explored the performance of the traditional VM deployments and the comparison with the use of Linux Containers. It used a suite of workloads that stress CPU, memory, storage, and networking resources. KVM was used as a representative hypervisor where as Docker is for the Container manager. It gives the result which shows that Containers result in equal or better performance than VMs in numerous cases. Both VMs and Containers require tuning to support I/O intensive applications. It also discussed the implications of the performance results for future cloud architectures. No tests were done on encryption and decryption security features in the testing environment [24].

It measured and analyzed the performance of the three open source VM monitors i.e. KVM, OpenVZ and Xen and KVM. The VM monitors adopt the para-virtualization, full virtualization and the Container based virtualization respectively. It measured them both as a black box and white box about their macro performance and micro performance on the virtualization of system operation. The experimental data showed some of the valuable information for designers and also provided a comprehensive performance understanding for the users. The performance measure and analysis was only tested in VMs [13].

Again, it was proposed about the hybrid virtualization which ran the para-virtualized guest in the hardware-assisted VM Container to take advantage of both. Experiment results indicate that the hybrid solution outweighs origin para-virtualization by nearly 30% in memory intensive test and 50% in micro benchmarks. Meanwhile, It was compared with the origin hardware assisted VM, hybrid guest owns over 16% improvement in I/O intensive workloads. It also showed that instead of hybrid virtualization, Container virtualization can be used to improve performance greatly [29].

V. OBJECTIVE OF THE PRESENT WORK

According to the several literature reviews, it has been found that it requires testing based on Quality of Service(QoS), network performance and security evaluations which was conducted by the benchmark tests like process virtualizations, memory virtualizations, disk and network virtualizations benchmark tests.

VI. IMPLEMENTATION OF PERFORMANCE SIMULATION

A VM impersonates a complete server. In a virtualized server environment, each VM guest includes a complete OS along with any binaries, libraries and drivers, and then the actual application on top of the Container. For the implementation of VMs in the system, Oracle virtual box was used and for the implementation of the Container in the system, Docker was installed which is an open source Containerization tool. After installing the software, the test methodologies were designed and conducted. The summary of the evaluation is in the following parts:

Quality of service evaluation. The QOS of VM and Container are used to implement the flow control for each network interface. This shows the fine-grained management support for VM and Container.

Network performance evaluation. The overhead caused by the software switching was determined. It compared the functions with respect to the internal network of the individual system respectively.

Security evaluation. Here it evaluated the level of file encryption with standard cryptographic functions in both the VM and Container.

The work details can be summarized as follows:

- VM was installed in the host Ubuntu system. The VM was formulated by Oracle virtual box running a guest Ubuntu 14.10 OS.
- The Containerization was realized by installing the open source Docker application which allows to build own Containers from scratch. In the Docker, it downloaded a light base Ubuntu image upon which it builds the Container with suitable dependencies to run all the simulation tests.
- Then the environment was set up for the simulation process by installing all the relevant libraries, dependencies and binaries both in the VM environment and the Container environment.
- All the tests were run on the above machines and compared in a precise and cautious environment.

A mid range system was used to carry out the tests. The system was an Intel core i5 processor running at 2.50 GHz. The Container was equipped with 3 GB of RAM with 500GB hard disk. The Container had a Ubuntu 14.10 image beforehand. The Container was running on Docker. The VM was equipped with 3GB of RAM with 100GB of hard disk. The guest OS in the VM was a Ubuntu 14.04. The VM was running on Oracle virtual box. Table I describes the detail system specifications for installing both the VM and Container.

TABLE I. SYSTEM SPECIFICATION

Component	**Hardware Configuration**	**Software Configuration**
Host	Intel i5 4GB RAM 500GB 7200RPM	Ubuntu14.04
VM	Intel i5 3GB RAM 100GB 7200RPM	Ubuntu14.04
Container	Intel i5 3GB RAM 500GB 7200RPM	Ubuntu14.10

VII. EXPERIMENTAL SET UP

Both the VM and Container were set up for running sets of tests for performance analysis in their respective machines through Phoronix test suite v 5.6.0. Several tests were conducted and has been summarized in each parts.

A. AIO Stress and Ram-speed benchmark test

This is an asynchronous I/O benchmark which was created by SuSE organization. The test provisions a part of the local or global thread memory and a read write job is done and the result is the net time taken and the memory bandwidth consumed. It assigns a single thread which performs 10 GB of writes to main memory in the present experiments. In Fig. 3 the result of the AIO stress is illustrated. This test created a set of global and local thread memory in both the VM and the Container. That thread memory was heavily optimized for carrying out read and write operation. The AIO stress benchmark initiated random write operation in the local and global thread memories and the write speed was compared with the Container and VM. From the result graph the result is given in MB/s and it has been concluded that the Container fared better than VM.

Here the system's memory performance is benchmark tested. The results of this test disclosed a basic harmony on all memory access operations. Arithmetic operational speeds are same, but the speed of copy is lower. The memory read/write bandwidth show the performance of a typical memory architecture: level I cache, level II cache and main memory. This test of system memory operation shows the results of memory operation due to system virtualization.

In Fig. 4 the ram speed benchmark carried out memory access operation with respect to arithmetic operational speeds. The integer memory access operation was done on the level 1 and 2 cache and main memory and the access speed was recorded in MB/s. From the result graph the Container's memory speed was marginally better than the VM.

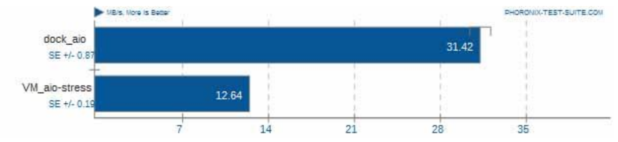

Fig. 3. AIO stress benchmark

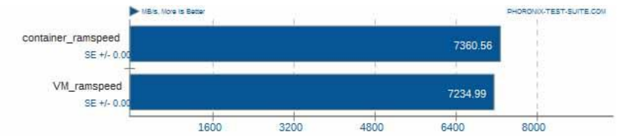

Fig. 4. Ram speed benchmark

B. IO ZONE and Network loop-back test

Here the benchmark tests the file-system performance and hard disk drive. The test creates special files on the disk and subsequently a read write job is conducted. It was tested how a single threaded process performed random read/write operation on the system files. It ran IOZONE test in different environments. The file read and writes bandwidth of both VM and Container was measured against each other. IOZONE has a two-level disk and according to the CPU's priority, the assignment of i/o bandwidth is done in accordance to the Container's priority. This test also checked the "page flipping speed".

The Fig. 5 chalks out the result graph of the IOZONE test. The average disk read write performance was recorded with the same set of files created both in Container and VM with a single threaded process. The file read write bandwidth was recorded in MB/s and from the given result graph the Container had a much better bandwidth than the VM.

The loop back network adapter performance is tested which uses a micro-benchmark to measure its TCP performance precisely [7] [9] [10]. To test network performance, it has used two tests, iperf and tbench. Tbench use the simulated network filesystem and sends TCP data over it and the throughput is measured.[15] Iperf is used to calculate the max UDP or TCP bandwidth among networks. The iperf runs as server on one host and client on another host network.

Fig. 6 gives the result of the network loopback test. Here the TCP performance was measured. Tbench sent TCP data worth 10GB over the network loopback and time taken to transfer the data was measured. From the given result graph the result is given in seconds and in this case the VM took less time as compared to Container. Thus in network loopback test the VM did better than Container.

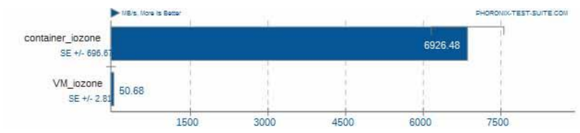

Fig. 5. Iozone benchmark

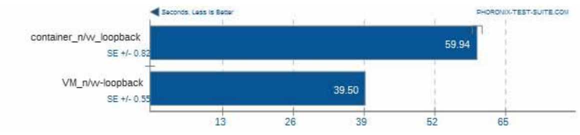

Fig. 6. Network loopback benchmark

C. Build Apache and Build PHP test

This measures the time taken to build and render an Apache HTTP server. This test is revolves on the two tier Web Server and Database. It uses the Apache web server 2.2 and the SQL database.

To emulate the actual scenario of web app environment, it installs, PHP version of RuBBoS benchmark and other Apache extension were installed. The RuBBoS data was stored in the SQL db. The benchmark was again installed in the current test settings. RuBBoS requires connection timing and maximum request rates.

Fig. 7 shows the result of apache server test. The test records the time to build, compile and run an Apache HTTP web server built in MYSQL database. From the result graph it can be seen that the Container takes less time to build the server than the VM. So in this test the Container did better than the VM.

The time was taken to build and render php5 server along with Zend engine. The Zend framework is an open source framework for building web applications that gets implemented in php5. The Zend community provides a php5 stack which is used to build a server in compliance to the model view framework.

Fig. 8 gives the result of the time to build the server in both the Container and VM. The standard php5 code gets to run in the zend framework in order to make the server go live which can run any web applications. From the result figure, it is seen that the Container takes less time to compile, build and run the server than the VM. Thus the Container does better in web application server setup than the VM.

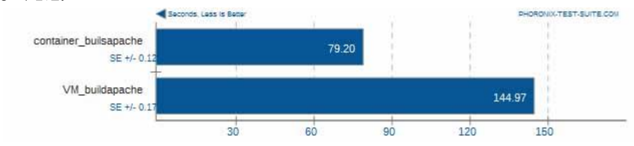

Fig. 7. Build Apache benchmark

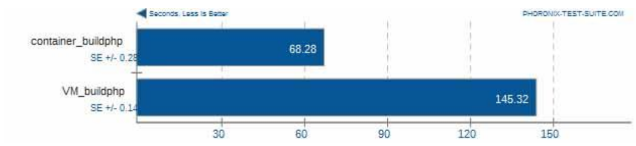

Fig. 8. PHP benchmark

D. Apache and G-crypt test

This test uses the Apache benchmark program. This test profile precisely measures how many requests per second a given system can sustain while carrying out around 1,000,000 requests with 100 requests being carried out concurrently.

Fig. 9 shows the apache benchmark test. This test measures the apache web server results. The apache web server is an efficient and secure server and it provides standard HTTP services in compliance with current HTTP standards. The test measures number of static web page requests that the apache server can handle. From the test result graph it can be seen that the Container can handle more number of HTTP requests per second than the VM. Thus the Container did better in web page serving than VM.

This is a libgcrypt's integrated benchmark with the CAMELLIA256-ECB cipher and with 100 repetitions. Camellia is specifically a symmetric key block cipher and in this case it has a block size of 256 bits. Its security keys are in compliance with the AES(Advanced Encryption Standards). Camellia is a fiestel cipher with 18 or 24 rounds. A logical transformation is applied every 6 rounds. It uses 8x8 bit S-box with input output affine transformation. Also a key whitener is used. Finally the diffusion layer uses linear transformation. In Fig. 10 the result for encryption and decryption the Camellia cipher is given. Here the encryption and decryption was performed with 100 specific repetitions and the time to complete the process was recorded. The less time a system takes the better it is security wise. Here the VM took less time as compared to the Container and thus the VM fared better over the Container.

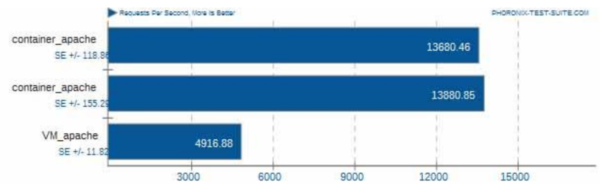

Fig. 9. Apache benchmark test

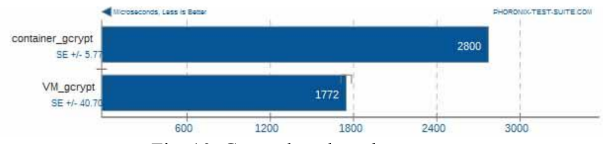

Fig. 10. Gcrypt benchmark

E. Blake2 and 7-zip compression test

BLAKE2 is a high-performance crypto alternative to MD5 and SHA-2/3.This is a benchmark uses the blake2's binary. Blake2 is a cryptographic hash function and is basically an improved version of SHA-3. It repeatedly combines an eight word hash value with around 16 message word to obtain the next hash value. Blake uses 32 bit words which are used for computing hashes up to 256 bits long and 64 bit words for computing hashes 512 bits long. The core block algorithm of the blake2 combines the 16 words of input with 16 working variables but only 8 words are saved between the blocks.

Fig. 11 shows the result of this cryptographic test and measures the time taken to cipher and decipher the input. From the figure both the Container and VM scored equally and took the same time in this blake2 cryptographic test.

We chose the 7-Zip benchmark which is an open source LZMA compression system. It measures the CPU performances. It finds out a system's performance with respect to millions of instructions per second (MIPS). This test expands and contracts bulk of data and accordingly calculates the performances. 7 zip works on multi threads on the cores of processing and gives the result in the form of MIPS.

Fig. 12 shows the benchmark result for the 7zip compression test and according to the figure the compress speed test was recorded in MIPS and Container had a greater MIPS value than VM. Thus Container had a better 7 zip compression result than VM.

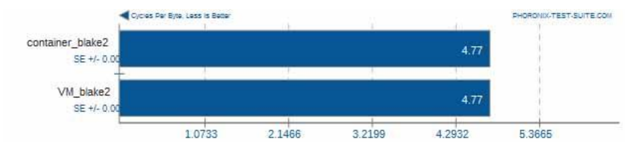

Fig. 11. Blake2 benchmark

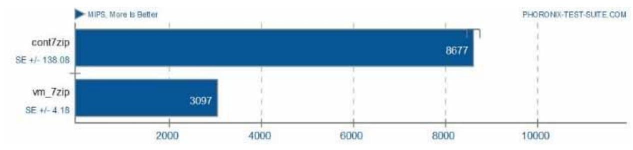

Fig. 12. 7-zip benchmark

F. OpenSSL test

We chose the OpenSSL benchmark to test the implementation of open source of the SSL (Secure Socket Layer) and TLS (Transport Layer Security) protocols along with general purpose cryptography library. Its main function is security and implements the cryptographic functions and other security features. It contains a set of encryption tools which a majority of web servers use. OpenSSL consists an assortment of different cryptographic algorithms like ciphers, cryptographic hash functions and public key cryptography.

In Fig. 13 the result of OpenSSL test is illustrated. It measures the number of OpenSSL certificates it has signed in the web. From the figure, the Container signed more number of OpenSSL certificates than VM thus Container is better than VM in OpenSSL test.

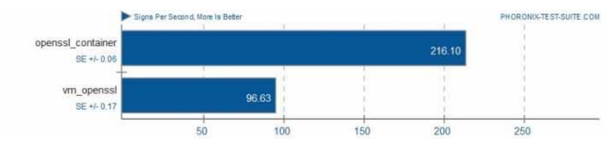

Fig. 13. OpenSSL benchmark

VIII. TEST RESULTS

All the above mentioned tests were investigated and compared both in a VM and in the Container. The resultant graphs were compared on the performance of both the test case machines. The conducted tests gave us a conclusion that Containers, mostly performed better over VMs. The mentioned Table II gives us an insight about the performance analysis of Containers and VMs based on the tests which have been performed.

In performance and efficiency wise the VM slightly suffered overhead compared to Containers. As far as security is concerned the VM had a slight edge over Containers as the VM had complete isolation. Even in cryptographic tests the VM performed just better than the Container. Thus the security aspect of Container leaves a future scope of further research. On storage and memory front, the VM carved out a big chunk of storage space from the host machine but Container was dynamic in nature as far as memory allocation was concerned and took less storage. The VM had a higher level of isolation than the Container as deduced from various tests performed. In networking, VM was supported by the hypervisor and in case of Containers it had sockets, signals but not a lot of advanced features was available. On boot up time Containers had a huge advantage over VM. Table II summarizes the performance of VMs and Containers deduced from the above mentioned tests.

TABLE II. PERFORMANCE COMPARISON

Attributes	VM	Container
Guest Operating System	Each VM runs a top of hypervisor and kernel loaded into its own memory region.	All guests share the same kernel. Kernel image is loaded in its physical memory.
Performance/ Efficiency	Suffers light over head as the machine instructions get translated from guest to host OS	Almost native performance as compared to the underlying host OS.
Security	Complete Isolation	Isolation using namespaces
Storage	Takes more storage	Take less storage as the base OS is shared
Isolation	Higher level of Isolation–Need special techniques for file sharing	Subdirectories can be transparently mounted and can be shared
Networking	Can be linked to virtual or physical switches. Hypervisor have their own buffers for IO performance improvement etc.	Leverage standards like IPC mechanisms like signals, pipes, sockets, etc. Advanced features like NIC not available
Bootup Time	Take a few minutes to boot up	Containers boot up in a few seconds as compared to VMs

IX. CONCLUSION

The usage of Containers in PaaS is becoming main stream and the Linux Containers are becoming a de facto standard. Some of the Cloud providers are using Containers to provide a fine-grained control over resource sharing. However, Both VM and cloud Container have undergone a big leap in their technological advancement since their creation. Though it has been deduced that VM has a fine performance there are challenges it faces that is over head performance. These overheads definitely impact the overall performance. It was also observed that the Containers themselves have almost no overhead, but still it has its own gigs. Containers gives overhead with considerably high packet rates. VM is operated individually enough to be expressed like it provides new computer to user. Because of this, it is easy to manage and apply the policy of system, network, user and security. Also, user can use various virtualization OS regardless of Host OS. In this work, we have implemented the evaluation environments and conducted different tests on the VMs and Containers simultaneously. The result shows that Containers score marginally over virtualization entity. However, Container has its own sets of short comings in the areas of security and isolation as it gives low security isolation than hardware virtualization.

REFERENCES

[1] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, I. Brandic, “Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility”, Future Gener. Comput. Syst., VOl.25, pp. 599–616, 2009

[2] Xudong Yang ,Ruozhou Yu, Jun Huang, QiangDuan, Yoshiaki Tanaka and Yan Ma “QoS-aware Service Selection in Virtualization-based Cloud Computing”, APNOMS, 2012

[3] FrancoCallegati, Walter Cerroni, Chiara Contoli, Giuliano Santandrea “Performance of Network Virtualization in Cloud Computing Infrastructures: The OpenStack Case” , IEEE 3rd International Conference on Cloud Networking (CloudNet), 2014

[4] David Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes” , IEEE Computer Society, VOl 1, pp: 81-84, 2014

[5] Aya Omezzine, Sami Yangui, Narjes Bellamine, Samir Tata, “Mobile service micro-Containers for Cloud environments”, IEEE 21st International Conference WETICE, 2012

[6] Ernest Artiaga, Jonathan Mart, Toni Cortes, “Better Cloud Storage Usability Through Name Space Virtualization”, IEEE/ACM 6th International Conference on Utility and Cloud Computing, 2013

[7] QiangDuan, Yuhong Yan, and Athanasios V. Vasilakos, “A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing”, IEEE Transactions On Network And Service Management, VOL. 9, NO. 4, DECEMBER 2012

[8] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, “Virtualization vs Containerization to support PaaS”, IEEE International Conference on Cloud Engineering, 2014

[9] Zongjian He, Guanqing Liang, “Research and Evaluation of Network Virtualization in Cloud Computing Environment”, 3rd International Conference on Networking and Distributed Computing, 2012

[10] Shuping Peng, Reza Nejabati, and Dimitra Simeonidou, “Role of Optical Network Virtualization in Cloud Computing ”,Journal OPT. COMMUN. NETW.VOL. 5, NO. 10, OCTOBER 2013

[11] Kyoung-TaekSeo, Hyun-Seo Hwang, Il-Young Moon, Oh-Young Kwon, Byeong-Jun Kim., "Performance Comparison Analysis of Linux Container and Virtual Machine for Building Cloud," Advanced Science and Technology Letters Vol.66, pp.105-111, 2014 http://dx.doi.org/10.14257/astl.2014.66.25

[12] Ming Zhao, Renato J. Figueiredo., "Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources", 2nd International Workshop on VTDC, pp.1-8, 2007

[13] JianhuaChe, Yong Yu, Congcong Shi, Weimin Lin., "A Synthetical Performance Evaluation of OpenVZ, Xen and KVM", IEEE Asia-Pacific Services Computing Conference, 2010

[14] Michael Seibold, Andreas Wolke, Martina Albutiu, Martin Bichler, Alfons Kemper, Thomas Setzer., "Efficient Deployment of Main-memory DBMS inVirtualized Data Centers", IEEE Fifth International Conference on Cloud Computing, 2012

[15] Ryan Shea, Jiangchuan Liu., "Performance of Virtual Machines Under Networked Denial of Service Attacks: Experiments and Analysis", IEEE SYSTEMS JOURNAL, VOL. 7, NO. 2, JUNE 2013

[16] Ryota Kawashima., "vNFC: A Virtual Networking Function Container for SDN-enabled Virtual Networks", IEEE Second Symposium on Network Cloud Computing and Application,2012

[17] David Breitgand, Amir Epstein, Alex Glikson, Assaf Israel, Danny Raz., "Network Aware Virtual Machine and Image Placement in a Cloud", 9th CNSM and Workshops,2013

[18] Abhishek Gupta, Laxmikant V. Kale, Filippo Gioachin, Verdi March, Chun HuiSuen, Bu-Sung Lee, Paolo Faraboschi, Richard Kaufmann, DejanMilojicic, "The Who, What, Why, and How of High Performance Computing in the Cloud", IEEE International Conference on Cloud Computing Technology and Science, 2013

[19] Tuan Dinh Le, SeongHoon Kim, Minh Hoang Nguyen, Daeyoung Kim, Seung Young Shin, Kyung Eun Lee , Rodrigo da Rosa Righi, "EPC Information Services with No-SQL data store for the Internet of Things" , IEEE International Conference on RFID, 2014

[20] John Paul Walters, Andrew J. Younge, Dong-In Kang, Ke-Thia Yao, Mikyung Kang, Stephen P. Crago, Geoffrey C. Fox., "GPU Passthrough Performance: A Comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL Applications", IEEE International Conference on Cloud Computing, 2014

[21] Yongen Yu, HongboZou, Wei Tang, Liwei Liu., "A CCG Virtual System for Big Data Application Communication Costs Analysis", IEEE International Conference on Big Data, 2014

[22] Paul Rad, Van Lindberg, Jeff Prevost, Weining Zhang, Mo Jamshidi., "ZeroVM: Secure Distributed Processing for Big Data Analytics", World Automation Congress, 20I4

[23] Yongen Yu, Hongbo Zou, Wei Tang, Liwei Liu, Fei Teng, "FlexTuner: A Flexible Container-based Tuning System for Cloud Applications", IEEE International Conference on Cloud Engineering, 2015

[24] Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio., "An Updated Performance Comparison of Virtual Machines and Linux Containers", IEEE International Symposium on Performance Analysis of Systems and Software, 2015

[25] Yuda Wang, Renyu Yang, TianyuWo, Wenbo Jiang, Chunming Hu., "Improving Utilization through Dynamic VM Resource Allocation in Hybrid Cloud Environment", 20th IEEE International conference on Parallel and Distributed Systems, pp. 241-248, 2014

[26] Xiaofei Liao, Hai Jin, Haikun Liu., "Towards a green cluster through dynamic remapping of virtual machines", Future Generation Computer Systems Vol. 28,pp. 469–477, 2012

[27] Hai Jin, Li Deng, Song Wu, Xuanhua Shi, Hanhua Chen, Xiaodong Pan., "MECOM: Live migration of virtual machines by adaptively compressing memory pages" Future Generation Computer Systems, Vol. 38, pp. 23–35, 2014

[28] G. Calarco, M. Casoni., "On the effectiveness of Linux Containers for network virtualization", Simulation Modelling Practice and Theory, Vol. 31, pp.169–185, 2013

[29] Qian Lin, Zhengwei Qi, Jiewei Wu, Yaozu Dong., Haibing Guan ,"Optimizing virtual machines using hybrid virtualization", The Journal of Systems and Software, Vol. 85, pp.2593– 2603, 2012

[30] Francesco Longo, “Design and integration of the Containers inspection activities in the Container terminal operations”, Int. J. Production Economics,Vol. 125, pp.272–283, 2010

[31] Internet-1: http://www.eweek.com/cloud/slideshows/10-quick-facts-about-docker-Container-virtualization.html (Accessed on 21st June , 2015)