Is Container-Based Technology a Winner for High Performance Scientific Applications?

Theodora Adufu, Jieun Choi, Yoonhee Kim
Dept. of Computer Science,
Sookmyung Women's University,
Seoul, Korea
theoadufu@gmail.com, (jechoi1205, yulan)@sookmyung.ac.kr

***Abstract*—High Performance Computing (HPC) applications require systems with environments for maximum use of limited resources to facilitate efficient computations. However, these systems are faced with a large trade-off between efficient resource allocation and minimum execution times for the applications executed on them. Also, deploying applications in newer environments is exacting. To alleviate this challenge, container-based systems are recently being deployed to reduce the trade-off. In this paper, we investigate container-based technology as an efficient virtualization technology for running high performance scientific applications. We select Docker as the container-based technology for our test bed. We execute *autodock3*, a molecular modeling simulation software mostly used for Protein-ligand docking, in Docker containers and VMs created using OpenStack. We compare the execution times of the docking process in both Docker containers and in VMs.**

Keywords*—*High Performance Computing (HPC), Container-based virtualization, Docker, Hypervisor-based virtualization (HPV), Cloud Computing, OpenStack.

I. Introduction

In recent years, virtualization technologies have been adopted to support efficient scientific computations and high performance applications. Correspondingly, there have been diverse Cloud Management Platforms (CMP) which provision and manage various computing resources. At the infrastructural level, platforms like OpenStack [1], AmazonEC2 [2], and Nimbus [3] are mostly used to provision and manage both private and public cloud platforms with their processor, storage, and network resources. These aforementioned middleware systems developed on the basis of hypervisor (HPV) virtualization technology however, require the installation of Guest Operating Systems (Guest OS) for each virtual machine (VM) created. This approach requires memory resources and slows down overall execution times of applications. Containers on the other hand, do not require Guest OS thus are more light-weight compared to hypervisor-based virtualization technologies.

Using Docker container-based systems, we demonstrate that the light-weight feature of container-based virtualization compared to hypervisor-based virtualization reduces the overall execution times of HPC scientific applications due to approximately zero start-up time when launching containers. We also demonstrate that even though the most utilized resource in the Docker container-based system is main memory (RAM), Docker manages memory resources efficiently hence creating a stable environment for HPC applications. We claim that, for HPC applications which require real-time launching of resources, container-based systems are more suitable.

The rest of the paper is organized as follows: Section II explores some related works and Section III gives a brief introduction of hypervisor-based and container-based virtualization systems with highlights on some differences. Section IV gives brief details of Docker, OpenStack and Autodocking, in line with the test bed environments for this paper. The experiment and the results will be evaluated in Section V and the paper will be concluded in Section VI.

II. Related Works

Rajdeep [4], analyzes 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 which use ***cgroups*** for resource allocation, ***chroot*** for processes and ***namespaces*** for network and resource isolation, have an inherent advantage over VMs because of performance improvements and reduced start-up times.

Miguel [5], also conducts benchmark experiments to support the claim that for better resource sharing and user-oriented custom HPC environments, there is the need for an isolation layer without performance overheads. In another paper [6] by the same authors, performance and manageability of containers were experimented. From the results of both experiments, container-based systems performed better in processing, memory, disks and network tests than HPV systems, represented by Xen. Xen however had better isolation due to non-shared Operating System (OS). These experiments are however benchmark experiments and do not reflect the real performance of HPC applications in container-based virtualization systems when deployed.

Using standard benchmarks, Walter [7] also conducts performance evaluation experiments for VMware Server, Xen and OpenVZ. Results reveal that, OpenVzs operating system-level virtualization system has the best overall performance.

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (NRF-2013R1A1A3007866)

Memari [8], proposes a new network protection approach, Honeynet, based on container-based virtualization. This increased security and system stability by isolating compromised honeynets from other honeypots on the server. Container-based virtualization also enables fast deployment and easier maintenance of honeypots for low physical resource utilization. During the experiments, honeynets implemented using LXC container-based virtualization proved more stable than those using KVM and VMware.

Stephen [9] demonstrates that container-based systems are more suitable for usage scenarios that require high levels of isolation and efficiency such as HPC Clusters. Resource isolation and security isolation were examined for some container-based systems and benchmark experiments were conducted to support their claims. Their results indicate that container-based systems perform two times better for server-type workloads than hypervisor-based systems.

III. HYPERVISOR-BASED VIRTUALIZATION VS CONTAINER-BASED VIRTUALIZATION

Virtualization technologies first adopted by IBM to support high-level software sharing have improved server utilization by allowing multiple Operating Systems to run in isolated virtual machines (VM) hosted on a single physical server [10].

For traditional virtualization, the hypervisor manages the computing resources of the host machine. Memory resources in particular can be overcommitted in hypervisor-based systems since most processes (or Virtual Machines) do not exhaust all of their allocated memory. Thus through cloud management platforms, VMs are provisioned and managed on each host machine such that users are able to access more computing resources than is physically available.

On the other hand, container-based virtualization sometimes known as Operating System-Level Virtualization, is deployed to improve resource sharing. It allows for multiple isolated user-space instances and facilitates the creation and maintenance of multiple environments customized according to each users needs. Container-based systems create abstractions for guest processes directly without the need to host a Guest OS thus reduces the overhead of creating a new VM with a Guest OS for each application.

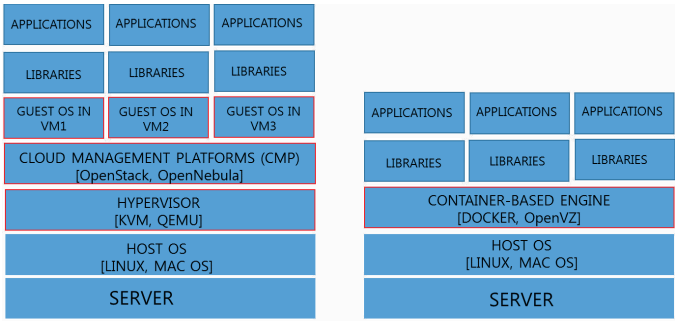

Fig. 1: Architectures for Hypervisor-based virtualization and container-based virtualization. Updated from [11].

Processes (or containers) are isolated using namespaces which have a lower performance overhead. Without the hypervisor, CMP and Guest OS layers as seen in Fig. 1, containers are lighter and hence easier and faster to launch. Also, they are likely to use less memory and disk-space allowing for multiple containers to be launched on a single host.

IV. DOCKER, OPENSTACK AND MOLECULAR DOCKING

A. DOCKER

The Docker container-based technology, has a single shared operating system with multiple isolated user space instances running on a single host [12]. Docker uses *cgroups* to allocate resources such as CPU, Memory and I/O control to containers [13]. To maximize memory resources utilization, Docker engine uses swapping technique thus, swap memory can also be explicitly allocated using *cgroups*.

B. OPENSTACK

Currently, hypervisor-based cloud platforms managed by Cloud Management Platforms (CMP) like OpenStack, provision and manage computing resources for large-scale scientific experiments.

OpenStack, has multiple services which enhance the efficient management of cloud resources. This includes the Nova Compute which is used for the creation of new virtual machines or instances. The resources of each VM such as RAM, disk and vCPUs, created using Nova Compute is defined by using default or user-defined install templates or flavors [14]. Memory and CPU can be overcommitted on compute nodes enabling access to more computing resources though this affects the performance of the instances [15]. This notwithstanding, OpenStack is widely deployed for the provisioning and management of cloud resources for HPC applications.

C. AUTODOCKING (MOLECULAR DOCKING)

High Throughput Computing(HTC) and Many Task Computing(MTC) applications usually consist of millions or billions of tasks with relatively high per task execution time. Tasks may be small or large, uniprocessor or multiprocessor, compute-intensive or data-intensive. For our experiments, we select *autodock3*, a molecular modeling simulation software, from a wide range of scientific computing applications to represent CPU intensive jobs. Molecular docking refers to a Structure-Based Drug Design (SBDD) computation through which the prediction of the binding modes of small molecule ligands within the active site of a target protein models is done [16]. The development of drugs such as HIV protease inhibitors is based on such structure-based design and screening strategies requiring the management of high volumes of data [17]. Docking experiments are compute-intensive and require fast and reliable access to memory.

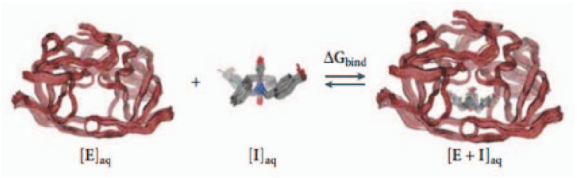

Fig. 2: Illustration of binding of inhibitor Dmp323 to HIV protease. Picture taken from [17].

Docking experiments are compute-intensive and require fast and reliable access to memory. They also require the management of high volumes of data [17]. For typical docking systems, docking is repeated several times in order to obtain consistent and accurate results from the analysis of the predicted energy values. For this experiment also, a single docking process will be executed repeatedly in each of the containers and VMs created. However, only one ligand and protein pair will be used for docking as a miniature representation of large-scale docking processes. The docking application used in this experiment is the suite of *autodock3* tools [18] compatible with Linux Operating System.

V. Experiments and Evaluation

The molecular modeling simulation software, *autodock3*, is executed in containers and VMs hosted on two Intel(R) Core(TM) i7 CPU 950 @ 3.07GHz server machines. Each machine has a total RAM size of 30GB and runs Ubuntu 14.04 Trusty Tahr Operating System. The base image selected for creating the Docker containers and for the Guest OS in the VMs, is Ubuntu 14.04 to ensure fair comparisons.

The first set of experiments take into account the start-up times of both the containers and VMs as this is very essential in measuring the overall execution times of applications and especially for resource scaling decisions. Each of the containers and VMs was created independently using Ubuntu 14.04 Server image with a memory allocation of 1GB and 1 vCPU. The relevant working files and directory were preloaded unto the base images used for the experiments.

The results in Fig. 3 show that in the Docker container, it takes an average of 176 seconds to launch a container and execute the *autodock3* while it takes an average of 191 seconds to launch a VM instance and run *autodock3*. Autodocking in the Docker containers is executed over a relatively shorter time period than in the VMs as a result of longer start-up times associated with launching new VM instances. This is because contrary to the VMs, Docker does not have to start-up Guest OS in the containers. Many large-scale elastic scientific applications require resource auto-scaling thus seek to launch instances in the least time possible. Also, for large scientific workflow applications where each task requires the creation of a new container, Docker containers seem more suitable.

The graph in Fig. 3shows the average execution times of running *autodock3* in 5 containers and VMs each.

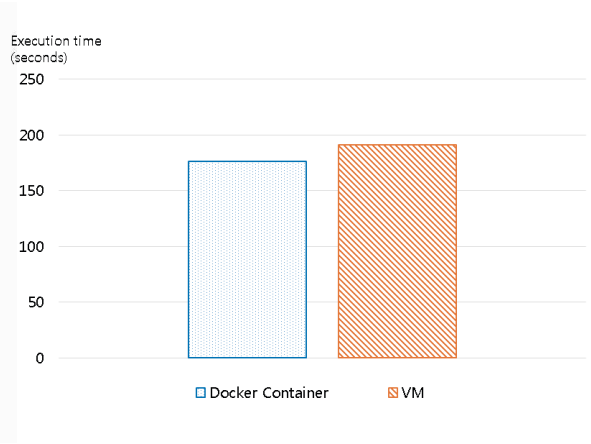

Fig. 3: Comparing total execution times of autodock3 in Docker Containers and VMs.

The second set of experiments seek to measure how memory allocations in Docker containers compares with similar allocations in VMs. Similarly, each of the containers and VMs was created independently using Ubuntu 14.04 Server image but with a memory allocation of 3GB. In order to measure the effect of memory allocations on the execution times of the docking process without the influence of start-up times, the containers and VMs were launched before the Autodocking processes were executed. For instance, for the total memory allocation of 12GB, 4 containers and 4 VMs were launched and *autodock3* was executed in parallel on each of them. The average time for executing *autodock3* in all running containers and VMs is compared in a graph.

Fig. 4 displays the execution times for parallel execution of *autodock3* in Docker containers and VMs when the total amount of main memory allocated is 12GB, 24GB, 36GB and 48GB respectively.

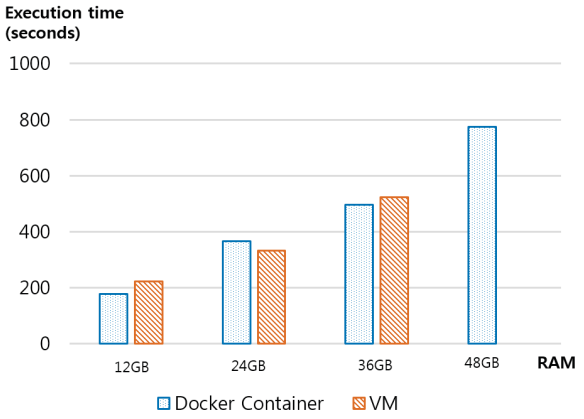

Fig. 4: Total execution times of autodock3 in Docker Containers and VMs for different memory allocations

The results show that when the amount of allocated memory

is 12GB, *autodock3* takes a relatively shorter time to execute in both Docker containers and VMs. The average execution time of the docking process increases with increase in allocated memory as a result of multiple processes running simultaneously thus requiring more resources.

For total allocated memory of 24GB however, *autodock3* performs better on VMs than in Docker containers. This reveals that there are some performance overheads in Docker containers due to physical resource limitations. When more than available physical memory resources are allocated to the containers and VMs however, Docker containers perform better than VMs for a total memory allocations of both 36GB and 48GB. For 48GB, *autodock3* executes in parallel in all Docker containers but fails to run in the VMs. This is due to Openstacks available memory overcommit ratio of 1.5 RAM. Following this ratio, only VMs with a total memory allocation of about 45GB can be launched in parallel on our host machine thus the failure for 48GB of memory allocation. This reveals another strength of Docker containers which is conducive for traditional parallel scientific HPC applications.

VI. Conclusion and future works

In this paper, we demonstrate that during the execution of scientific applications, overall execution times for container-based virtualization systems are less than in hypervisor-based virtualization systems due to differences in start-up times.

We run *autodock3*, a molecular modeling simulation software in containers and virtual machines, in similar host environments and compare the execution times for the docking processes. Results from the experiments show that, Docker manages memory resources more efficiently even when more than available physical memory resources are allocated to running instances.

From our results, container-based systems are more efficient in reducing the overall execution times for HPC applications and have better memory management for multiple containers running in parallel. We conclude that Container-based systems are more suitable for HPC applications.

In the future, we will investigate a framework for running scientific applications with different characteristics, in cluster of container-based virtualization systems and evaluate the optimal environment for efficiently executing the applications.

References

[1] OpenStack, http://www.OpenStack.org

[2] Amazon Web Services, http://aws.amazon.com

[3] Nimbus cloud, http://www.nimbusproject.org/

[4] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, Virtualization vs Containerization to support PaaS Cloud Engineering (IC2E), 2014 IEEE International Conference, pp. 610- 614 (2014)

[5] Miguel Gomes Xavier, Marcelo V. Neves, Cesar A. F. De Rose, Performance Evaluation of Container-based Virtualization for High Performance Computing Environments Parallel, Distributed & Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, 2013, pp. 233-240 (2013)

[6] Miguel Gomes Xavier, Marcelo V. Neves, Cesar A. F. De Rose A Performance Comparison of Container-based Virtualization for MapReduce Clusters Parallel, Distributed & Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, 2014, pp. 299-306 (2014)

[7] J. P. Walters, Vipin Chaudhary, Minsuk Cha, Salvatore Guercio Jr., Steve Gallo, A Comparison of Virtualization Technologies for HPC, Advanced Information Networking & Applications, 2008. AINA 2008. 22nd International Conference on 2008, pp. 861-868 (2008)

[8] Nogol Memari, Shaiful Jahari B. Hashim, Khairulmizam B. Samsudin, Towards virtual honeynet based on LXC virtualization Region 10 Symposium, 2014 IEEE, pp. 496-501 (2014)

[9] Stephen Soltesz, Herbert Potzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson, Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors, EuroSys '07 Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pp. 275-287 (2007)

[10] Yan Junhao, Lv Aili, Research on the Application of Virtualization Technology in High Performance Computing Electrical and Electronics Engineering (EEESYM), 2012 IEEE Symposium on 24-27 June 2012, pp. 386 388 (2012)

[11] How is this different from virtual machines?https://www.docker.com/whatisdocker/

[12] James Turnbull, The Docker Book, August 4, 2014, (2014)

[13] Resource Management in a Docker,https://goldmann.pl/blog/2014/09/11/resource-management-in-Docker

[14] Flavors, http://docs.openstack.org/openstack-ops/content/flavors.html

[15] Overcommiting on compute nodes, http://docs.openstack.org/openstack-ops/content/compute_nodes.html

[16] Ocana K., Benza S., De Oliveira D., Dias J., Mattoso M., "Exploring Large Scale Receptor-Ligand Pairs in Molecular Docking Workflows in HPC Clouds", Parallel & Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International, pp. 536-545 (2014)

[17] Douglas B. Kitchen, Hlne Decornez, John R. Furr, Jrgen Bajorath, Docking and Scoring in Virtual Screening for Drug Discovery: Methods And Applications, Nature Reviews Drug Discovery, 3(11), pp. 93-949 (2004)

[18] Autodock, http://autodock.scripps.edu/