Using Docker in High Performance Computing Applications

Minh Thanh Chung, Nguyen Quang-Hung, Manh-Thin Nguyen, Nam Thoai

Faculty of Computer Science and Engineering,
HCMC University of Technology, VNU-HCM
Ho Chi Minh City, Vietnam
Email: {51102040}@hcmut.edu.vn, {hungnq2,nmthin,nam}@cse.hcmut.edu.vn

***Abstract*—Virtualization technology plays a vital role in cloud computing. In particular, benefits of virtualization are widely employed in high performance computing (HPC) applications. Recently, virtual machines (VMs) and Docker containers known as two virtualization platforms need to be explored for developing applications efficiently. We target a model for deploying distributed applications on Docker containers, among using well-known benchmarks to evaluate performance between VMs and containers. Based on their architecture, we propose benchmark scenarios to analyze the computing performance and the ability of data access on HPC system. Remarkably, Docker container has more advantages than virtual machine in terms of data intensive application and computing ability, especially the overhead of Docker is trivial. However, Docker architecture has some drawbacks in resource management. Our experiment and evaluation show how to deploy efficiently high performance computing applications on Docker containers and VMs.**

***Keywords*-Docker, HPC, performance evaluation, Graph500, HPL, cloud computing.**

I. Introduction

The development of technology is recently being associated with the growth of problem size that we need to handle. The problem size is increasing exponentially and physical machine cannot support for a huge range of users on different operation environments simultaneously. That is one of reasons leading to the advent of cloud computing and virtualization technology. Virtualization contributes to solutions in providing resources and sharing computational spaces as well as storage spaces.

Recently, VMs and containers are considered as virtualization techniques. VMs are the backbone at the infrastructure layer supporting cloud services, such as Platform-as-a-Service (PaaS). Containers are the same in the purpose of use, but having different implementation and architecture. Functionalities of container are being employed in cloud computing field with many benefits for reducing overhead and providing lightweight characteristics. VMs tend to virtualize at the hardware abstraction level, but containers are the emulation at the operating system (OS) abstraction level. Following that, we carry out research on analyzing differences between these two platforms with advantages and disadvantages as well. Especially in Docker [1], it is a new platform supporting containers for shipping, running and developing applications. Based on the potentials of Docker container, our research explores a model to deploy distributed application on this infrastructure. When running HPC applications with the scalable problem size, the performance of Docker container is whether better or not than virtual machine. Furthermore, the architecture of Docker allows to share resources with the OS kernel, which is still doubted about the ability of data access inside containers. We use well-known HPC tools represented by High Performance Linpack (HPL) [2] and Graph500 [3] in order to evaluate the performance, where HPL solves a dense linear system to measure computing ability and Graph500 is a graph algorithm standing for data intensive applications. In measurement, we consider Graph500 as the standard of ability of data access, we name criterion of this ability data traceability. These benchmarks are performed with many different problem sizes and scenarios. Our evaluation highlights two criteria of performance of VMs and containers, especially in the measurement about the data traceability.

In this paper, we review the fundamental background of hypervisor-based and container-based technique in section 2. Then, we mention about related works in section 3 and state problems of containerization in section 4. Section 5 introduces a model considered as a strategy to deploy distributed applications in Docker containers. In section 6, we show different evaluations about the performance of VMs and Docker containers. Remarkably, Docker has impressive results of performance based on its architecture. Finally, we summarize our evaluations and propose future works in section 7.

II. Hypervisor-based and Container-based virtualization

There are two models that allow to deploy virtualized instances. They are hypervisor-based and container-based platform [4]. For VMs, hypervisor known as a layer deploys, allocates operation space of instances. As the architecture of real computer, to operate a system we need an interaction between software and hardware. In particular, architected interfaces [5] contribute to a role in managing resources. This is an essential feature for VM construction. Three of important interfaces are instruction set architecture (ISA), application binary interface (ABI) and application programming interface (API).

978-1-5090-1801-7/16/$31.00 ©2016 IEEE

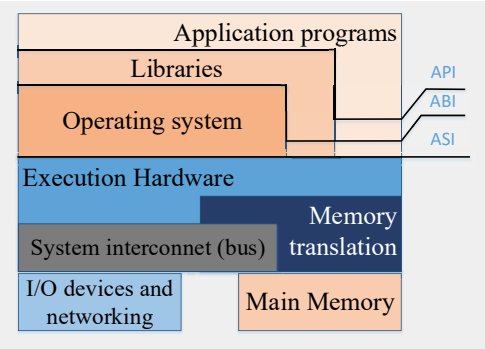

Fig. 1. Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API) [5]

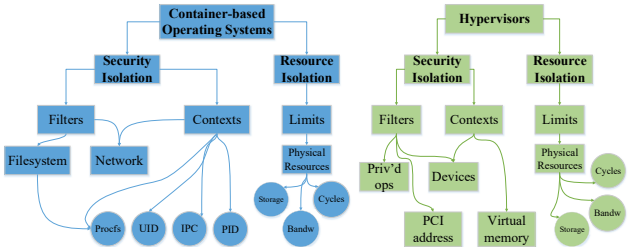

Fig. 2. Isolation Taxonomy of COS and Hypervisor-based Systems [12]

Figure 1 reveals computer system architecture with key implementation layers communicated via interfaces such as ISA, ABI, API. In virtual machine, a process or system running is $guest$ and the underlying platform supporting virtualized instances is $host$. Hence, they need a hypervisor [6] or virtual machine monitor (VMM) to manage, deploy VMs. From the perspective of the operating system and the application it supports, a generated virtual machine has a whole execution environment which can perform many processes simultaneously. It can allocate individual memory and I/O resources to the processes. The VMM has to emulate the hardware ISA that the guest software can execute.

In addition, figure 2 shows the differences in techniques used to implement isolation of hypervisor-based and container-based architecture. Hypervisor focuses on the isolation and security. VMs are constructed at the hardware abstraction layer such creating virtual devices, privileged instructions, virtual memory address spaces. To meet with the demand of large scale cluster in HPC applications, virtual machine have benefits in such systems, from the following aspects [7] :

- **Complete isolation and security**: Hypervisor supports an entire environment to each instances. A virtual machine can be ensured the operation space with the allocated resources for other guests. Every malicious code in guest OS crashes a virtual machine, which can recover because the resource integrity of the physical machine is controlled by VMM.
- **Ease of management**: A virtual machine may create or delete, start or shutdown much easier than a real machine. An administrator can observe VMs and additional information running inside each instance.
- **Customization**: We can modify not only the virtualized software, but also the emulated hardware before creating a guest. With the allocated physical resources, VMs can scale the specification of system (i.e., CPU, RAM), depending on user requirements.

In contrast to hypervisor-based platform, containerization allows to share OS image including a root file system, libraries and common files. As shows in figure 2, the container-based platform virtualizes at the OS abstraction layer. To reach the security isolation, containers use OS objects to define operation space, e.g., PIDs, UIDs, IPC. Similar to hypervisor, the isolation of containers is implemented by two models, $contexts$ and $filters$. The definition of $contexts$ means the separation of namespaces and $filter$ stands for the access controls. Because the virtualization at OS abstraction layer, containers need a $namespaces$ [12] to implement the isolation in this platform. Hence, at the view of host machine, a container seems to be a process when running a job. It is marked by a unique PID, UID, IPC, etc.

Remarkably, Docker [9] known as the container-based virtualization is becoming increasingly popular in cloud computing. As mentioned above, Docker is a platform supporting container for developing, shipping and running distributed applications. Without the hypervisor layer, Docker containers are deployed and controlled by Docker engine [9]. Based on the implementation technique, Docker has more lightweight, which deriving from three components consisting of Docker images, Docker registries and Docker containers. It allows to share the same OS kernel that containers start instantly and make more efficiently use of RAM. Especially, Docker makes use of union file system [10] to create images as well as make distributing Docker images faster and simpler with the feature is called $copy - on - write$ [11]. This is one of benefits to meet the demand of scalability in cloud computing.

III. Related Works

The fact that virtualization technologies have explored and utilized for a long time [5] [13]. In practice, VMs are playing a vital role in cloud computing field [7]. However, the overhead is one of drawbacks of VMs and we need to enhance, when using them for HPC applications. Recently, Docker is a new platform supporting containers and they are explored the foundation of architecture [14] priliminarily with some results of performance at basic scenarios [15]. In this paper, we consider much more about some problems related to scalability of the amount of the generated containers, e.g, the changes of performance when deploying HPC applications on one virtualized instance and more virtualized instances. For the potential of containerization architecture, Docker is opening a revolution in cloud computing. Docker is developed and applied its functionality to practical applications, especially in high performance computing. Our research compares the implementation techniques between VMs and Docker containers. We introduce the model to deploy distributed applications on

Docker platform. However, the efficiency of Docker in HPC is still questionable. Several approaches have been pursued to survey both virtual machine and Docker with benchmarks [4] [8] [15]. Our evaluation focuses on two criteria consisting of computational capacity and data traceability which can reflect one of advantages in architecture sharing resource with the host kernel. Those criteria are represented by two well-known HPC tools, High Performance Linpack [16] and Graph500 [17]. Finally, we conduct analysis that bases on performance results and architectures.

IV. Problems in containerization technology

Virtualization technologies is being applied in every areas. However, to employ them efficiently, we need to consider virtualization types, architectures and strong as well as weak points carefully. There are two virtualization architectures represented by two platforms as mentioned in section II, namely VMs and Docker containers [18]. Therefore, the differences between of VMs and Docker containers on each specific purpose need to be explored.

From the architecture of Docker and its advantage, the problem arisen is how to deploy distributed applications on this platform. The application running on Docker container is whether different or similar to VMs. While each instance of virtual machine has an entire OS and isolate the resource space, Docker container shares resources with the host kernel.

Through analyzing the architecture of VMs and containers, we focus on two performance criteria, computing ability and data traceability. Docker container is still doubted if it is suitable on HPC system. That is the problem when we deploy HPC applications on Docker. The performance is questionable when increasing a large range of generated VMs or containers on a whole system.

V. A model for deploying distributed applications on Docker

Many types of applications can be configured with different approaches depending on each virtualized architecture. For example, VMs as the hypervisor-based instances have full components emulated by hypervisor layer, e.g., hardware, OS, libraries. Hypervisor has to deploy an entire OS and filesystem in each virtual machine. This results in the overhead of emulating OS and libraries when generating a large range of VMs. While the advantage of VMs is isolation, its disadvantage is overhead when running distributed applications. This feature is also one of problems that developers have to consider in PaaS field [14]. Based on the container-based architecture, Docker is a platform supporting containers that can share the same OS kernel and related libraries. In further, Docker containers can share common files because their images are constructed from layered filesystems [19]. When running a job, each container is assigned a unique PID, it can be observed equivalently as a process at the view of host machine. Through these characteristics of Docker, we deploy applications that share the same dependencies, essential libraries under the host machine. This method is available for solving scalable problems and portable computations because we can reduce remarkably the overhead, when comparing to VMs.

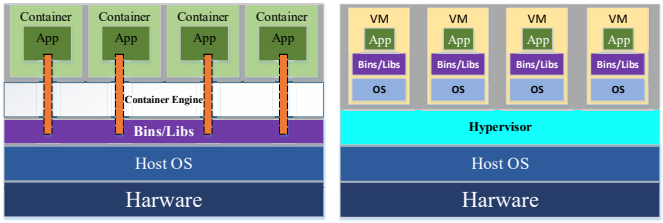

Fig. 3. Model for deploying distributed applications on Docker and Virtual Machine

Figure 3 shows the model which we deploy Docker container to run applications. Normally, VMs provide a complete environment which supports multiple users as well as applications. VMs emulate the hardware and full OS along with individual libraries. Hence, we have the same way to configure distributed application on VMs and host system. We need to install and configure applications, libraries inside each virtual machine to execute as a cluster. In contrast to VMs, we exploit the sharing ability of Docker with host OS kernel to deploy applications. Each Docker container does not need to set up a whole OS or image with related libraries, they can share the same binaries and libraries during executing. Dockers architecture uses client-server model with three main components including: Docker image, Docker registry and Docker container. A container is created from a Docker image and it then creates a read-write layer on top of image using union file system (UnionFS) [10]. Union file system allows Docker image divide into many layers. When containers run, UnionFS creates a writable layer on the top and we use this layer to update into a new image. Typically, the libraries and environment variables under host are mounted to this new image, meanwhile, running containers. There is only our application on Docker container, the required libraries can share with host OS. Our applications run on Docker container with these advantages that make system more lightweight and faster. This is a model which we propose to deploy distributed applications on Docker container, especially in HPC applications.

VI. Experimental results and analysis

With the model of deploying distributed applications on Docker containers (figure 3), we investigate experiments about practical applications. Concretely, we use computing intensive and data intensive applications, namely High Performance Linpack (HPL) [16] and Graph500 [20]. Our target is to test the efficiency of HPC applications deployed on Docker containers by the sharing model (figure 3) and compare to VMs.

A. Testing environment

In this paper, we set up an evaluation environment with the limitation of unimportant services to reduce overhead. This means that there are merely pure OS and related dependencies

TABLE I
SCENARIOS OF CONFIGURATION IN PERFORMANCE EVALUATION

The number of instances	Virtual machine (vCPU, vRAM, execution processes)	Docker (execution processes)
2	16, 32, 16	16
4	8, 16, 8	8
8	4, 8, 4	4
16	2, 4, 2	2
32	1, 2, 1	1

on the verified system. We use native performance as a standard to evaluate the overhead of virtualized environment. Objectively, we propose many benchmark scenarios with different configurations on Docker containers and VMs. The resources allocated in VMs or containers have to saturate with the resources of system under test, e.g. RAM, cores.

We implement all of tests on Intel System with two compute nodes. There are 16 physical cores totally (along with Hyper Threading technology) with Intel Xeon CPU E5-2670 @ 2.6GHz and 64 GB of RAM. Between two compute nodes, the network equipped under system is 1Gbps Gigabit Ethernet. For consistency, the OS that we use is CentOS 7 64-bit 3.10.0 on both of physical and virtual environment. In further, VMs and Docker containers are deployed by the latest version such as QEMU 2.4.0.1/KVM and Docker 1.7.1 respectively. Our testbeds use full of resources (e.g. CPU, RAM) of physical system with the problem size is equivalent to the capacity of system under test. First, we target the reasonable sizes of problems which our system can obtain the best performance on each environment. Second, we benchmark on many configurations of VMs and Docker containers with the increasing number of generated instances. This allows to observe the changes of performance when we run more than one virtual instance. The VMs or Docker containers are deployed with different quantities. We have 64GB of RAM and 32 logical cores which the real system can allocate for virtual environment. Table I shows the configurations of VMs and Docker containers with the difference of each case, being the number of generated instances among the values of CPU, RAM and execution processes.

B. Docker container with computing intensive application

Computing intensive application is represented by HPL benchmark. This is a portable implementation of Linpack benchmark. The problem of HPL is to generate, solve, check and time the random dense linear system of equations. This is a familiar problem which its size can scale up or down. HPL uses double-precision floating point arithmetic and portable routines for linear algebra operations, message passing. Afterwards, the benchmark counts floating-point operations per second and returns performance results. To run an application as HPL benchmark on distributed system, we need three main parts including math library, message passing interface (MPI) and HPL package.

According to section V, we configure HPL inside each Docker container, but math library and MPI are mounted from the host OS. Concretely, we use OpenBLAS [22] and

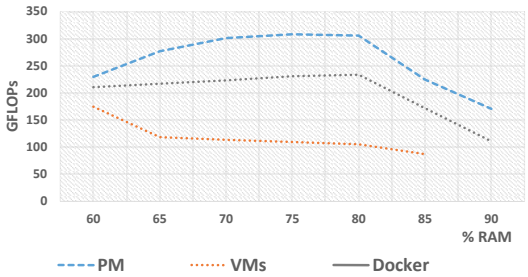

Fig. 4. HPL benchmark with different problem sizes in PM, VMs and Docker containers

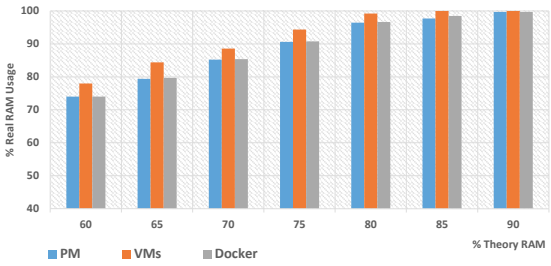

Fig. 5. The real cost of memory usage in PM, VMs and Docker containers at the view of PM

OpenMPI [23] representing math library and MPI, which they are installed under host. As figure 3, because each Docker container performs as a process inside host OS, all of libraries and dependencies being necessary for HPL are mounted directly to container. This facilitates to create multiple containers on a node. For VMs, HPL needs to set up along with OpenBLAS and OpenMPI inside each instance. Hence, the overhead proliferates over the increasing number of generated VMs.

The problem of HPL is matrix size and it is one of parameters directly affecting to computing performance. First, we construct a scenario with a rage of matrix sizes between VMs and Docker containers. Matrix sizes stretch from 60% to 90% of RAM in system under test. In this case, figure 4 shows the efficiency of Docker container when running HPC application based on our deployment method in section V. The native performance is the best result and it is considered as a based case, accounting for roughly 308 GFLOPs in figure 3. HPL performance of Docker container is better than virtual machine, especially, the matrix size occupies from 75% to 80% of RAM is suitable for Docker. By contrast, VMs obtain the better performance in matrix size being smaller than 65% of RAM. Additionally, HPL results cannot be returned if the size is over 85% of RAM. Having analyzed the real cost of memory that physical system has to pay, figure 5 compares the real memory utilization between VMs and Docker containers when performing HPL benchmark. The percentage of RAM usage on VMs is larger than containers, leading to the overhead increases. Therefore, when the matrix size scales up, the performance gets lower.

Second, the scenario targets more about the number of

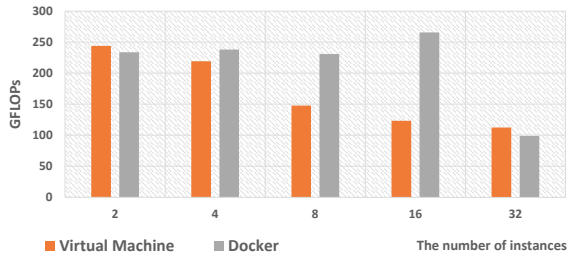

Fig. 6. HPL results in virtual machine and Docker container with different configurations

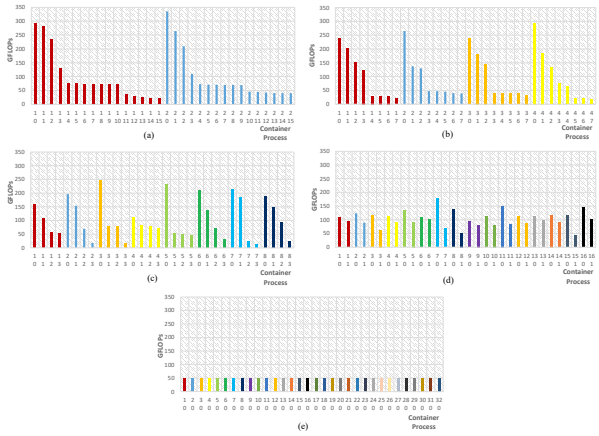

Fig. 7. Workload distribution on Dockers cases including: (a) 2-container case, (b) 4-container case, (c) 8-container case, (d) 16-container case, (e) 32-container case

generated instances of Docker and virtual machine. As the table I, we configure the specification of Docker containers and VMs to observe the performance. Overall, figure 6 shows the performance of Dockers situations seems to be higher than virtual machine in almost configurations. For increasing the number of created instances, while HPL results of VMs decreases gradually, Docker has the best performance with 16-container cluster. To analyze this result, one of reasons that we survey is the distribution of workload inside each container. Figure 7 shows that 16-container cluster distributes CPU workload fairer than others. When running many processes on each container, Docker has a drawback of controlling computing workload. Furthermore, we execute HPL with full of computing resources and the number of containers is maximum, which leading to the computing ability is dropped, e.g., the 32-container case. This makes Docker consume much more overhead in managing as well as distributing containers and sharing resources. Through the evaluation above, distributed applications are deployed on Dockers architecture with sharing libraries, this results in many advantages about overhead but we have to consider to the resource management of Docker such as CPU cycles.

C. Docker container with data intensive application

We use Graph500 to represent the data intensive application. Graph500 is a benchmark for supercomputers using large-scale

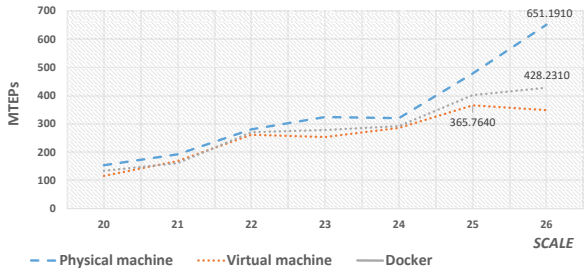

Fig. 8. Graph500 results with different sizes on physical machine, virtual machine and Docker container

graph to evaluate performance. The data traceability is returned by two phases. Phase 1 is undirected graph construction [20] and phase 2 is Breadth-First Search [17] to traverse the generated graph. Both phases are timed and the final result is TEPS meaning traversed edges per second. The problem size of Graph500 is graph size which we can tune for each specific system. There are five problem classes defined by [20], toy (17GB), mini (140GB), small (1TB), medium (17TB), large (140TB) and huge (1.1PB). They reflect to a parameter called $SCALE$. In practice, the graph is created by Kronecker [24] and developers have to adjust the graph size based on two parameters: $SCALE$ and $Edgefactor$ (which stands for the number of edges calculating by (1)).

$$M = E * 2^{SCALE} \tag{1}$$

where M is the number of edges, E is $Edgefactor$ and 2^{SCALE} is the total number of vertices. We need to set up Graph500 with MPI to run the benchmark. As well as HPL benchmark, we configure Graph500 inside each Docker container and share the libraries under the host kernel.

The first scenario is to run Graph500 in a range of graph sizes. Our testbeds are performed with $SCALE$ under 26 and $Edgefactor$ is default value (being 16). Figure 8 shows Graph500 between physical machine, VMs and Docker containers with the graph sizes stretch from $SCALE$ being 20 to $SCALE$ being 26. The difference is clear when Graph500 runs the large $SCALE$, e.g. 24, 25, 26. Overall, the data traceability of Docker is higher than virtual machine. Docker containers can run large scale, while the problem size is limited on VMs. One of reasons for this is surveyed similar to HPL benchmark, as figure 7 above. The overhead for VMs is larger than Docker containers when the graph size increases. This leads to the Graph500 performance of VMs is low.

The second scenario also performs on many configurations of VMs and Docker containers. As figure 9 shows, when increasing VMs, the Graph500 results which runs on the number of these VMs decrease almost as a linear line. Because the hypervisor has to deploy more necessary binaries and related libraries, entire guest OS. Without hypervisor [9], the Docker's architecture allows containers to retrieve data faster. By contrast, increasing number of Docker containers does not increase the overhead. Figure 9 also reveals that the 16-container case achieves the best performance of Graph500.

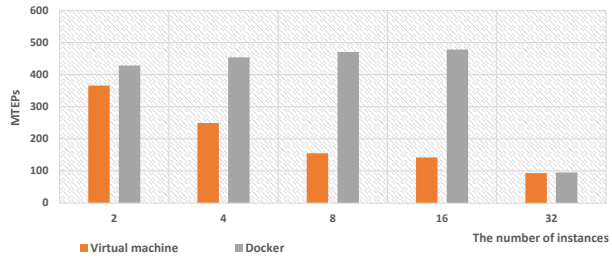

Fig. 9. Comparison the performance of Graph500 that runs on VMs and Docker containers with the increasing number of generated instances

When we create many containers on system, Docker Engine only needs to manage the resources in term of data traceability that need to be used. Containers can start instantly and make more efficient use of RAM. Especially, Docker containers share the images constructed from layered filesystems, therefore, they can share making disk usage and common files efficiently. Similar to HPL results (figure 6), when we create 32 containers, Docker has the drawback in controlling resources because of sharing the OS kernel. That is one of reasons that we need to investigate more about the algorithm of managing resources on Docker container.

VII. Conclusion and Future Work

Docker container or lightweight technology is becoming a widespread platform used in cloud computing field. In this paper, our research shows that VMs and Docker containers have both positive and negative factors. Thus, we have to consider about the target of utilization and the feature of application type running on them. While VMs have a strong point about the isolation criterion, Docker containers have numerous benefits in reducing overhead because the architecture allows to share the OS kernel. As the Docker's architecture, we exploit these characteristics to execute applications efficiently.

Our evaluation shows that the utilization of VMs and Docker containers obtains many advantages about portability, convenience and scalability. However, we need to pay attention about the problem sizes, application types and system limitations. For example, Docker is more suitable than virtual machine about data intensive applications.

In the future, we explore the transmission capacity in the cluster system, which is formed by VMs and Docker containers. Then, we investigate a formula to predict approximately the maximum quantity of VMs and Docker containers that the given system can generate. This future work will support the resource scheduler in allocating VMs and containers.

Acknowledgment

This research was conducted within the Studying and developing computing platform based on Docker and OpenStack technique sponsored by TIS (IT Holding Group).

References

[1] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.

[2] J. J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: past, present and future," *Concurrency and Computation: practice and experience*, vol. 15, no. 9, pp. 803–820, 2003.

[3] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, 2010.

[4] R. Morabito, J. Kjallman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 2015, pp. 386–393.

[5] J. E. Smith and R. Nair, "The architecture of virtual machines," *Computer*, vol. 38, no. 5, pp. 32–38, 2005.

[6] J. Hwang, S. Zeng, F. Y. Wu, and T. Wood, "A component-based performance comparison of four hypervisors," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 269–276.

[7] W. Huang, J. Liu, B. Abali, and D. K. Panda, "A case for high performance computing with virtual machines," in *Proceedings of the 20th annual international conference on Supercomputing*. ACM, 2006, pp. 125–134.

[8] R. Dua, A. R. Raja, and D. Kakadia, "Virtualization vs containerization to support paas," in *Cloud Engineering (IC2E), 2014 IEEE International Conference on*. IEEE, 2014, pp. 610–614.

[9] (2014) Docker homepage. [Online]. Available: https://www.docker.com/

[10] C. P. Wright and E. Zadok, "Kernel korner: unionfs: bringing filesystems together," *Linux Journal*, vol. 2004, no. 128, p. 8, 2004.

[11] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*. IEEE, 2015, pp. 342–346.

[12] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 275–287.

[13] S. G. Soriga and M. Barbulescu, "A comparison of the performance and scalability of xen and kvm hypervisors," in *Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition*. IEEE, 2013, pp. 1–6.

[14] C. Pahl, "Containerization and the paas cloud," *IEEE Cloud Computing*, no. 3, pp. 24–31, 2015.

[15] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[16] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. (2012) Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. [Online]. Available: http://www.netlib.org/benchmark/hpl/

[17] K. Ueno and T. Suzumura, "Highly scalable graph search for the graph500 benchmark," in *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. ACM, 2012, pp. 149–160.

[18] T. Adufu, J. Choi, and Y. Kim, "Is container-based technology a winner for high performance scientific applications?" in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. IEEE, 2015, pp. 507–510.

[19] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[20] D. Bader, J. Berry, S. Kahan, R. Murphy, J. Riedy, and J. Willcock, "The graph 500 list: Graph 500 reference implementations," *Graph500. http://www. graph500. org/reference. html (2 Febuary 2012)*, 2010.

[21] D. Ziakas, A. Baum, R. A. Maddox, and R. J. Safranek, "Intel® quickpath interconnect architectural features supporting scalable system architectures," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 1–6.

[22] Z. Xianyi, W. Qian, and Z. Chothia, "Openblas," *URL: http://xianyi. github. io/OpenBLAS*, 2014.

[23] M. Open, "Open source high performance computing," 2012.

[24] J. Leskovec, D. Chakrabarti, J. Kleinberg, and C. Faloutsos, "Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication," in *Knowledge Discovery in Databases: PKDD 2005*. Springer, 2005, pp. 133–145.