J Inf Process Syst, Vol.14, No.6, pp.1398~1404, December 2018
https://doi.org/10.3745/JIPS.01.0033

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Proposal of Container-Based HPC Structures and Performance Analysis

Chanho Yong*, Ga-Won Lee*, and Eui-Nam Huh*

Abstract

High-performance computing (HPC) provides to researchers a powerful ability to resolve problems with intensive computations, such as those in the math and medical fields. When an HPC platform is provided as a service, users may suffer from unexpected obstacles in developing and running applications due to restricted development environments and dependencies. In this context, operating system level virtualization can be a solution for HPC service to ensure lightweight virtualization and consistency in Dev-Ops environments. Therefore, this paper proposes three types of typical HPC structure for container environments built with HPC container and Docker. The three structures focus on smooth integration with existing HPC job framework, message passing interface (MPI). Lastly, the performance of the structures is analyzed with High Performance Linpack benchmark from the aspect of performance degradation in network communications under Docker.

Keywords

Container, Docker, High-Performance Computing, Singularity

1. Introduction

High-performance computing (HPC) services have provided an infrastructure to run computationally intensive jobs in the science and math fields. As users request more complex and varied environments for running and developing HPC programs, HPC service provider is finding it hard to deal with their needs, such as installing specific libraries or HPC software [1]. Although users want to use the modules, some modules are hard to be installed due to possible conflicts with existing software or unexpected impacts on HPC infrastructure [2].

Virtualization can be a solution to meet specific requirements, so HPC users can build their own environments freely. However, hypervisor-based virtualization techniques, such as kernel-based virtual machine (KVM) or Xen, have been regarded as inappropriate because of performance loss in the process of virtualization. The HPC container is considered a good example that successfully achieves integration with HPC platforms in operating system (OS) level virtualization, bringing low performance overhead [3]. However, rich functionalities of Docker container, such as effective isolation and extensibility, can be necessary in order to manage an HPC platform more flexible. Therefore, we propose

Manuscript received May 28, 2018; accepted July 29, 2018.
Corresponding Author: Eui-Nam Huh (johnhuh@khu.ac.kr)
* Dept. of Computer Science and Engineering, Kyung Hee University, Suwon, Korea (alice_k106@naver.com, {gawon, johnhuh}@khu.ac.kr)

three types of HPC structure using the HPC container and Docker, which enable container technology to integrate with message passing interface (MPI) framework. We also evaluate performance to validate the proposed structures by deploying a High-Performance Linpack (HPL) cluster in Singularity and Docker. As a result, we found that prevalent overlay networks in Docker, such as swarm mode and etcd overlay network, cause performance degradation from the aspect of network communications. Contributions of this paper are proposal for container-based HPC service structures considering integration of MPI framework, and the performance analysis for guidance on choosing a proper HPC structure.

2. Background

2.1 Message Passing Interface

MPI is a standard that describes exchange of information in distributed and parallel processing. MPI defines an abstract interface to receive and send messages between MPI processes, and thus, programs that comply with MPI standard can be portable and compatible from one to the other [2]. Even though MPI processes can be launched independently by 'mpiexec' or 'mpirun' commands, most HPC services run MPI program indirectly using a resource manager and scheduler installed on HPC platform for batching and queueing jobs. In this procedure, MPI processes are managed by a process manager with a corresponding process management interface (PMI) to control and communicate with MPI processes. Since the separation of PMI and MPI allows MPI libraries to stay generic enough to be used with any PMI, integrating MPI and an HPC platform requires PMI support in a built-in or indirect way.

2.2 Docker and HPC Containers

Docker is container management engine that provides virtualized space by utilizing cgroup, namespace, and chroot. In contrast to a virtual machine, which virtualizes hardware and requires an intermediate supervisor on top of the host OS, Docker is much more lightweight and composable because of the virtualization principle of sharing a host kernel [4]. Docker container and images consist of layered file system, which facilitates fast distribution of Docker images by not transferring existing layers in a host. However, Docker mainly focuses not on HPC workflow, but on establishing micro-service architecture for containerized applications. Not only is MPI integration with Docker container not officially supported, but Docker is also not compatible with the existing HPC scheduler and resource manager [5,6].

The HPC container is designed to integrate existing HPC workflow into OS-level virtualization while aligning itself to MPI framework in a user-defined environment. There are two HPC containers that are generally adopted by many communities: Shifter [1] and Singularity [7]. Shifter enables HPC users at the National Energy Research Scientific Computing Center (NERSC) to run a Docker image efficiently by constructing their own additional component. Singularity also supports compatibility with HPC system and MPI framework, and mainly focuses on portability to run the HPC container, regardless of the Linux distribution and execution environment.

3. Related Works

There has been a lot of research studying the introduction of OS-level virtualization into scientific and high-performance applications by utilizing Docker and HPC container, Shifter and Singularity container. Martin et al. [8] explored the feasibility of rkt container in a high-performance environment by conducting Graph500 and HPL benchmark tests, and they compared performance with commonly used container technologies, Docker and LXC (Linux container). Nguyen and Bein [9] implemented a ready-to-use solution to quickly deploy and use HPC environment for a distributed MPI program orchestrated with Docker swarm mode. Sparks [3] investigated the characteristics of the container ecosystem for an HPC workload and examined the execution time of container as well as numerical aerodynamic simulation (NAS) parallel benchmark of each container, rkt, Docker, Singularity, and runC. A lot of research into HPC proved that container technology can meet HPC requirements, by reason of low performance loss and feasibility, compared to traditional hypervisor-based virtualization.

4. Proposal of HPC Platform Structure Based on Containers

The goal of integrating containers into HPC environments is to avoid violating fundamental HPC systems such as the workload manager and MPI framework, while preserving the performance of native host. The HPC container is a natural way to maintain an existing HPC system because of its support for MPI framework interface. Although the HPC container accomplishes the eventual goals by complying with MPI interface and through compatibility with the HPC system, the functionalities of Docker represented by cgroup, layered file system, and network namespace are adapted as an alternative to the HPC container in order to meet various requirements of HPC service.

In contrast to the HPC container, which provides an interface compatible with the HPC workload manager, such as Slurm and Sun Grid Engine (SGE), Docker is designed to construct a micro-service architecture for scalability in a cloud domain. On account of this, Docker container is required to compose different architecture to provide HPC service for users. In this context, we propose three types of HPC structures that integrate container and HPC system using an HPC container and Docker. The structures focus on achieving interoperability and compatibility between containers and MPI implementations.

Fig. 1(a) represents integration of existing HPC workloads using an HPC container. The HPC container supports most of MPI library executions with HPC resource managers and scheduler. Each MPI process launched in an HPC container is managed by the HPC system in terms of process lifecycle and its communications. As the HPC container works in form of a plugin with additional commands or options like '--singularity-image' and 'shifter', the fundamental HPC structure remains unchanged. The proposed model achieves successful integration with the HPC system for container; nevertheless, there are still limitations in flexibility and extensibility for some aspects due to a lack of features in the HPC container itself. Depending on the requirements of users and its purposes, Docker substitutes for the HPC container shown in Fig. 1(a) with changes to the overall architecture.

The second and third structures were designed using approaches from the perspective of Docker. The second structure, which is shown in Fig. 1(b), shows characteristics similar to the structure in Fig. 1(a), because Docker container is regarded as a single MPI process or job. The difference is that workload

manager executes 'mpirun' or 'mpiexec' commands in each Docker container to launch parallel processes. In other words, the user executes MPI command in a common way, such as Bash shell, and it triggers utilizing a communications process, including Secure Shell Daemon in infrastructure as a service (IaaS) environments. In this way, job management is delegated to an externally implemented resource manager and scheduler, taking a straightforward structure for separation between container and HPC platform. However, because an HPC system that supports Docker container is not officially provided currently, it should be implemented independently by HPC service providers [10].

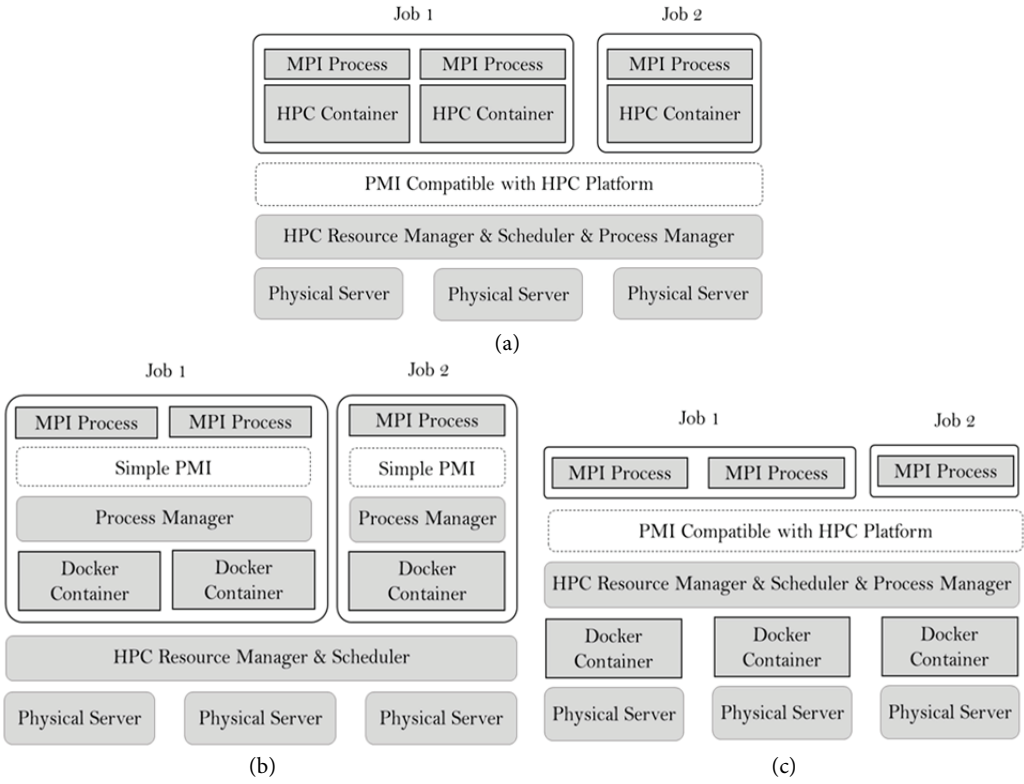

Fig. 1. HPC platform structure. (a) HPC container integration with existing HPC platform. (b) Docker case 1: container as MPI process of each job. (c) Docker case 2: container as HPC worker node.

The last structure also uses Docker container, as shown in Fig. 1(c), but HPC resource manager and scheduler manage each container as a node. All HPC components exist in Docker container, and each container is regarded as a consumable resource, like a virtual machine (VM) or native host. Therefore, the HPC system is implemented inside Docker container in the same manner as an existing HPC platform. The HPC platform is packaged as a Docker image to be deployed to an HPC cluster, while externally installed HPC system files may be mounted into containers to ensure the independence of its components. The third proposed structure has advantages in flexibly scaling HPC nodes by creating or deleting containers quickly. If users request guaranteed integrity of their own environments, MPI process is formed as a container, which requires container-in-container technology, such as Docker-in-Docker (dind) or Play-with-Docker (PWD).

5. Performance Evaluation

According to the proposed structures, Fig. 1(a) is categorized as HPC container, and Fig. 1(b) and 1(c) are categorized as Docker container. To examine the applicability of two container types to a practical HPC job from the performance aspect, an evaluation using Singularity and Docker was conducted. A benchmark cluster was deployed in both containers using HPL, which is one of the indicators used to rank Top500 supercomputers. HPL consists of distributed MPI processes to solve a random dense linear system in double-precision arithmetic.

Performance evaluation results from using Singularity version 2.3.1 and Docker version 17.03.0-ce are shown in Fig. 2. The test was set up under two hosts with CentOS version 7.2.1511 installed and equipped with 16 GB memory; 16 Intel Xeon E5-2620 CPUs at 2.10 GHz applied hyperthreading, which is equal to 32 logical CPUs. The network setup was configured under a 10 Gb network switch and an internal IP address to maximize bandwidth. For Singularity, the 'srun' command was used to launch MPI processes to each container, and for Docker, 'mpirun' was used. In both cases, CPU affinity was set by MPICH version 3.2 and Slurm version 17.02.6 to prevent unexpected performance loss in CPU scheduling. HPL.dat configuration in HPL benchmark is properly optimized to drive the best performance within provided hardware specifications.

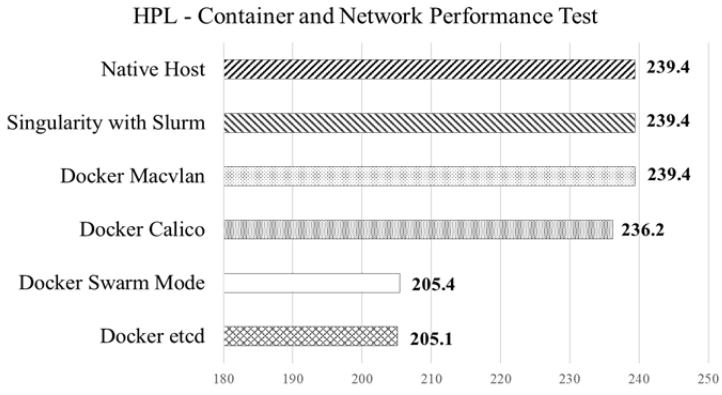

Fig. 2. Performance test with HPL benchmark.

An MPI process inside a Docker container needs a way to communicate with other containers located on different hosts, since each Docker container has Network Address Translation (NAT) IP address. The host network mode of Docker allows containers to have the same network configuration as the host, so containers in different hosts communicate by the hosts' IP addresses. However, this way regards each host as a consumable resource, and does not assume running multiple containers in a host. Thus, this paper covers alternatives for intercommunication between containers, overlay network of swarm mode, and distributed coordinators, such as etcd, Calico, and Macvlan.

In Singularity, performance is the same as native host because HPC containers do not utilize a network namespace, which is one of the reasons for performance degradation in network packet encapsulation. The Macvlan network of Docker also shows the same performance as native host. Macvlan allows the host machine to configure multiple L2 addresses on a single physical network interface, and it does not incur performance loss from a network aspect. However, Macvlan cannot be

used in a general environment because Macvlan for container communication between different hosts requires a specific network infrastructure such as network switch and router.

Performance degradation under Calico, showing an approximate 1.3% GFlops loss, is negligible when compared to swarm mode and etcd overlay network. Swarm mode and etcd overlay network showed considerable performance degradation of up to 14.2%. Even though swarm mode and etcd overlay network have such a performance loss, both have an advantage in that an additional agent in the Docker daemon is not required. Docker swarm mode supports physical hosts to compose logical clusters for pooling distributed resources, and etcd improves the potential extensibility of an application by exposing key-value information in the form of a distributed coordinator. Therefore, in Docker, it is necessary to adopt a suitable network method considering the characteristics of HPC infrastructure and network features.

6. Conclusion and Future Works

HPC systems get benefits from OS-level virtualization in building an efficient HPC service and platform. In order to standardize HPC platform structure based on container, this paper proposed three integration types that consider HPC workload and MPI framework in HPC container and Docker. The performance was examined by deploying the HPL benchmark in Singularity and Docker. Experiment results showed that Singularity has no performance loss, but in Docker, performance degradation occurred in swarm mode and etcd overlay network. Consequently, this result indicates that flexibility and network functionalities are a trade-off with performance.

In future works, provisioning HPC platform as a cloud service to provide on-demand scalability will be studied. Integrating cloud services with HPC services will enable HPC cloud users to deploy a high-performance infrastructure as they want it.

Acknowledgement

This work was supported by Korea Institute for Advancement of Technology grant funded by the Korea government (Ministry of Science and ICT) (Tech Commercialization Supporting Business based on Research Institute-Academic Cooperation system) and by the Ministry of Science and ICT, Korea, under the Information Technology Research Center support program (No. IITP-2018-2013-1-00717) supervised by the Institute for Information & communications Technology Promotion (IITP).

References

[1] D. M. Jacobsen and R. S. Canon, "Contain this, unleashing Docker for HPC," in *Proceedings of the Cray User Group*, Chicago, IL, 2015.

[2] M. de Bayser and R. Cerqueira, "Integrating MPI with Docker for HPC," in *Proceedings of 2017 IEEE International Conference on Cloud Engineering (IC2E),* 2017, pp. 259-265.

[3] J. Sparks, "HPC containers in use," in *Proceedings of the Cray User Group*, Redmond, WA, 2017.

[4] J. H. Huh and K. Seo, "Design and test bed experiments of server operation system using virtualization technology," *Human-centric Computing and Information Sciences*, vol. 6, article no. 1, 2016.
[5] L. Benedicic, F. A. Cruz, A. Madonna, and K. Mariotti, "Portable, high-performance containers for HPC," 2017 [Online]. Available: https://arxiv.org/abs/1704.03383.
[6] J. Higgins, V. Holmes, and C. Venters, "Orchestrating Docker containers in the HPC environment," in *High Performance Computing*. Cham: Springer, 2015, pp. 506-513.
[7] G. M. Kurtzer, V. Sochat, and M. W. Bauer, "Singularity: scientific containers for mobility of compute," *PloS One*, vol. 12, no. 5, article no. e0177459, 2017.
[8] J. P. Martin, A. Kandasamy, and K. Chandrasekaran, "Exploring the support for high performance applications in the container runtime environment," *Human-centric Computing and Information Sciences*, vol. 8, article no. 1, 2018.
[9] N. Nguyen and D. Bein, "Distributed MPI cluster with Docker swarm mode," in *Proceedings of 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC),* Las Vegas, NV, 2017, pp. 1-7.
[10] B. Gerofi, R. Riesen, R. W. Wisniewski, and Y. Ishikawa, "Toward full specialization of the HPC software stack: reconciling application containers and lightweight multi-kernels," in *Proceedings of the 7th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*, Washington, DC, 2017.

Chanho Yong https://orcid.org/0000-0002-6777-5500

He received a B.S. degree in Computer Science and Engineering from Kyung Hee University in 2017. Since March 2017, he has been in Computer Science and Engineering at Kyung Hee University as a master's degree candidate.

Ga-Won Lee https://orcid.org/0000-0002-6411-4467

She earned a Ph.D. degree in Computer Engineering from Kyung Hee University, South Korea, in 2013. At present, she is a research professor in the Department of Computer Science and Engineering at Kyung Hee University-Global Campus, Korea.

Eui-Nam Huh https://orcid.org/0000-0003-0184-6975

He received his bachelor degree in Computer Science and Engineering from Busan University, master degree of Computer Science from University of Texas, USA, and Ph.D. in Computer Science from the Ohio University, USA. He is a professor in department of Computer Science and Engineering, director of Real-Time Mobile Cloud Research Center and Dean of Information Services and Strategy in Kyung Hee University.