Performance evaluation of MEGADOCK protein–protein interaction prediction system implemented with distributed containers on a cloud computing environment

Kento Aoyama[1,2,a] Yuki Yamamoto[1,b] Masahito Ohue[1,c] Yutaka Akiyama[1,d]

Abstract: Container-based virtualization has begun to be introduced into large-scale parallel computing environments. In the bioinformatics field, where various dependent libraries and software tools need to be combined, the container technology that isolates the software environment and enables rapid distribution as in an immediate executable format, is expected to have many benefits. In this study, we employed Docker, which is an implementation of Linux containers, and implemented a distributed computing environment of our original protein–protein interaction prediction system, MEGADOCK, with virtual machine instances on Microsoft Azure cloud computing environment, and evaluated its parallel performance. Both when MEGADOCK was directly performed on the virtual machine and also when it is performed with Docker containers of MEGADOCK on the virtual machine, the execution speed achieved was almost equal even if the number of worker cores was increased up to approx. 500 cores.

Keywords: container-based virtualization, cloud computing, MPI, Docker, MEGADOCK, protein–protein interaction

1. Introduction

In the field of bioinformatics and computational biology, various software are utilized for research activities. Management of software environments such as dependent software libraries is one of the most challenging issues in computational research. Recently, as a solution to complication of the software environment, introduction of the container-based virtualization technology, which is an approach using virtualization technologies with lightweight and excellent performance, is advancing [1], [2]. Particularly in the field of genome research, pipeline software systems consisting of multiple pieces of software are commonly used, which tend to complicate the environment. For this reason, case studies have been reported, including those on environmental management and distributed processing using the container-based virtualization technology [3], [4].

In the container-based virtualization, a software execution environment, including dependent software libraries and execution binaries, is isolated as a container, and immediate software distribution as an executable format can be realized [5], [6]. This feature facilitates the management of the software environment and distribution, thus introducing new software. It has also been reported that the container-based virtualization performs better than the hypervisor-based virtualization, which is used to implement common virtual machines (VMs), and when properly configured, performs almost as well as running on a physical machine [7].

Container-based virtualization has grown in areas such as parallel distributed platforms on cloud environments because it enables rapid application deployment [8]. Although the introduction has not advanced in the computer environments at research institutes and universities due to the concern regarding performance degradation by virtualization, there is a tailwind through excellent benchmark results on parallel computing environments and application research case reports. Recently, the container-based virtualization technology has begun to be adopted in supercomputing environments. For the instance, the Singularity[9], a container implementation for HPC environments, is available on the TSUBAME 3.0 supercomputer of Tokyo Institute of Technology and AI Bridging Cloud Infrastructure (ABCI) of National Institute of Advanced Industrial Science and Technology (AIST), both of which are world's first-tier supercomputing environments in Japan [10]. From the above, correspondence to the container-based virtualization is urgent even in the bioinformatics and computational biology fields.

In this study, we focused on MEGADOCK [11], [12], a protein–protein interaction (PPI) prediction software, as an example of bioinformatics software, that can predict PPIs between various proteins by parallel computing. We introduced distributed processing on MEGADOCK using Docker [1], an implementation of the container-based virtualization, and then evaluated its computational performance on the Microsoft Azure public cloud computing environment [13] by comparing it with a simple parallel implementation with message passing interface (MPI) [14].

[1] Department of Computer Science, School of Computing, Tokyo Institute of Technology, Tokyo, Japan
[2] AIST-Tokyo Tech Real World Big-Data Computation Open Innovation Laboratory (RWBC-OIL), National Institute of Advanced Industrial Science and Technology, Ibaraki, Japan
[a] aoyama@bi.c.titech.ac.jp
[b] y_yamamoto@bi.c.titech.ac.jp
[c] ohue@c.titech.ac.jp
[d] akiyama@c.titech.ac.jp

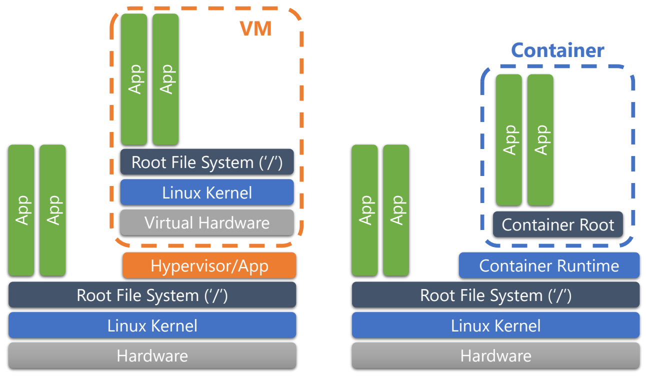

Fig. 1 Overview of virtualization technologies: hypervisor-based (left) and container-based virtualization (right)

2. Overview of container-based virtualization

There are two major concepts of virtualization approaches in the context of applications running on a cloud environment: hypervisor-based and container-based.

2.1 Hypervisor-based virtualization

In hypervisor-based virtualization, the virtual environment is provided by a higher-level "hypervisor" that further manages the OS (Supervisor) that manages the application (**Fig. 1**, left). The "virtual machine" (VM) widely used in general cloud environments is provided by the hypervisor-based virtualization that enables users to use various operating systems such as Windows and Linux OS as Guest OS, which is managed by a hypervisor running on the Host OS or hardware.

There are various types of implementations for the hypervisor-based virtualization, such as Kernel Virtual Machine (KVM) [15], Hyper-V [16] used in Microsoft Azure, XEN [17] used in Amazon Web Service, and VMware [18].

2.2 Container-based virtualization

In container-based virtualization, containers are realized by the isolation of the namespaces of user processes running on the host OS (Fig. 1, right). These virtualizations are mainly implemented by the namespace [5], one of the Linux kernel features. The namespace can isolate user processes from global namespaces to individual namespaces, and enable us to use different namespaces for mounting points, processes, users, networks, hostnames, etc. Therefore, users can touch an isolated application environment that is separate from the host environment. The container-based virtualization is sometimes called as kernels-sharing virtualization because the containers running on the same host commonly use the same kernel.

According to a previous study, performance overheads of a container in various aspects are smaller than those of the VMs because the resource management in containers is under the control of its host kernels [7]. Moreover, the data size of the container images tends to be smaller than that of the VMs. This offers a significant advantage on the application deployment. Docker [1] is widely used in cloud computing environments.

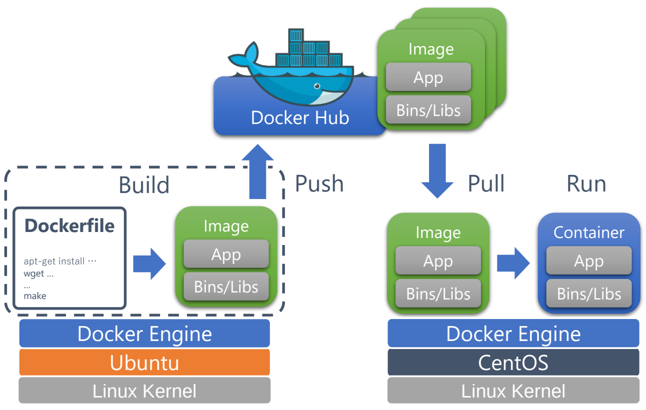

Fig. 2 Sharing a Docker container image via Docker Hub

2.3 Docker

Docker [1] is the most popular set of tools and platform for managing, deploying, sharing of Linux containers. It is an open-sourced software on the GitHub repository, operated by Moby [19] project, written in Golang, contributed by worldwide developers. There are several related toolsets and services of Docker ecosystems, such as Docker Hub [6], the largest container image registry service to exchange user-developed container images. Docker Machine [20] provides container environments to Windows and MacOS using a combination of Docker and the hypervisor-based approach.

2.3.1 Sharing container image via Docker Hub

A container image can include all the dependencies necessary to execute the target application: code, runtime, system tools, system libraries, and configurations. Thus, it enables us to reproduce the same application environment in the container as we build it, and deploy onto the machine with other specifications. Users easily share their own application environment with each other through uploading (push) of container images via the Docker Hub [6], the largest container image registry service for Docker containers, and downloading (pull) of the same container image onto a different machine environment (**Fig. 2**).

3. MEGADOCK

MEGADOCK is a PPI prediction software for a large-scale parallel computing environment, which was developed by Ohue *et al.* [11], [12]. MEGADOCK supports MPI, OpenMP, and GPU parallelization, and has achieved massive parallel computing on TSUBAME 2.5/3.0, K computer, etc. The MPI parallel implementation on Microsoft Azure [13] public cloud (MEGADOCK-Azure [14]) as well as the predicted PPI database MEGADOCK-Web [21] have been developed to promote the use of this software in more general environments.

Fig. 3 shows a diagram of the system architecture of parallel processing infrastructure on Microsoft Azure VMs using MEGADOCK-Azure.

3.1 MEGADOCK with container-based virtualization

The versatility of a software dependent environment with deployment/management problems and improvement of its execution performance are still pressing issues. The use of a cloud computing environment, such as the case of MEGADOCK-Azure,

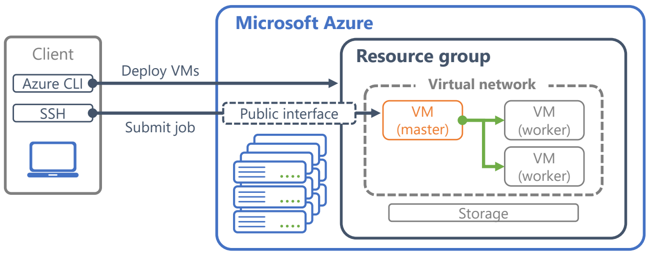

Fig. 3 System architecture of MEGADOCK-Azure [14]

which is based on VMs, is one of the solutions, but there are still concerns about the complexity to reuse the entire existing local computing resource, vendor lock-in problems, and performance degradation due to the hypervisor-based virtualization.

In this study, we tried to solve these problems by using Docker container, which is one of the implementations of the container-based virtualization. Introduction of container techniques into MEGADOCK has the following advantages:

- Docker containers are able to run on almost all environments over various cloud computing infrastructure using the same container image as well as on our local environments.
- The container-based virtualization approach generally shows superior performance than the hypervisor-based virtualization approach both in running and deploying.
- There are compatible container environments[9] available on several high-performance computing (HPC) environments such as TSUBAME 3.0 and ABCI supercomputers, such that it can even be a model of standard application package in HPC environments.

To maintain compatibility between different environments, we implemented the MEGADOCK system using Docker containers running on VM instances of Microsoft Azure by referring to the MEGADOCK-Azure architecture. Docker containers over different VM instances are connected on an overlay network using Docker networking functions and are able to communicate with each other using the MPI library. Thereby, we can deploy the Docker containers on VM instances to run the docking calculation of MEGADOCK, as shown in **Fig. 4**.

4. Performance evaluation

We measured the execution time and its parallel speed-up ratio of distributed MEGADOCK system by changing the number of worker processor cores of the VM instances in Microsoft Azure under the master-worker model on the MPIDP framework.

4.1 Experimental setup

We selected `Standard_D14_v2`, a high-end VM instance on Microsoft Azure. It is equpped with an Intel Xeon E5-2673 (16 cores) processor, 112 GB of RAM, and 800 GB of local SSD storage. The software environment is shown in **Table 1**.

The measured data were obtained from the result of the `time` command, and the median of 3 runs of the calculations was selected. To avoid slower data transfer time between nodes, all output results of docking calculation were generated onto local SSDs attached on each VM instance. On the Docker container case, to avoid the unnecessary performance degradation due to the layered file system of the container, all output files are stored to a data volume on local SSD which was mounted on the container.

Table 1 Software environment

	Virtual Machine	Docker
OS (image)	SUSE Linux	library/ubuntu
Version (tag)	Enterprise Server 12	14.04
Linux Kernel	3.12.43	N/A
GCC	4.8.3	4.8.4
FFTW	3.3.4	3.3.5
OpenMPI	1.10.2	1.6.5
Docker Engine	1.12.6	N/A

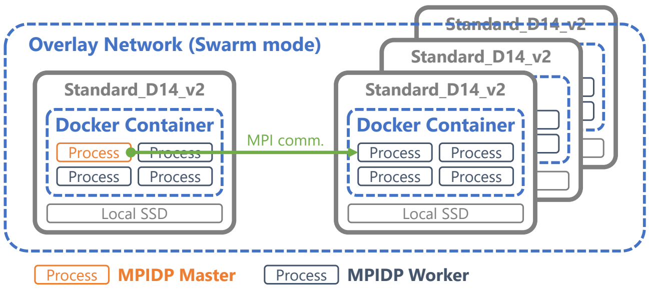

Fig. 4 Overview of overlay network using Docker Network

4.2 Dataset

Protein hetero-dimer complex structure data of the protein–protein docking benchmark version 1.0 [22] were used for performance evaluation. Whole 59 hetero-dimer protein complexes were used and all-to-all combinations of each binding partner ($59 \times 59 = 3{,}481$ pairs) were calculated to predict their possible PPIs.

4.3 Experiment result

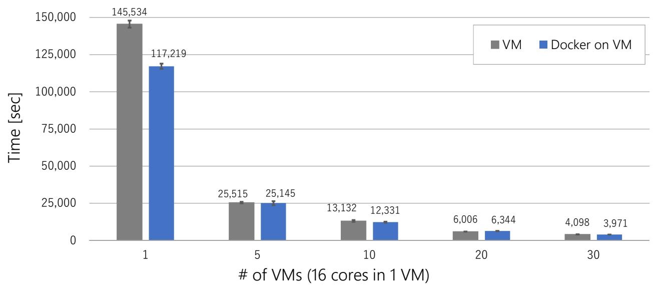

Fig. 5 Execution time comparison between MEGADOCK directly running on VMs and with Docker container

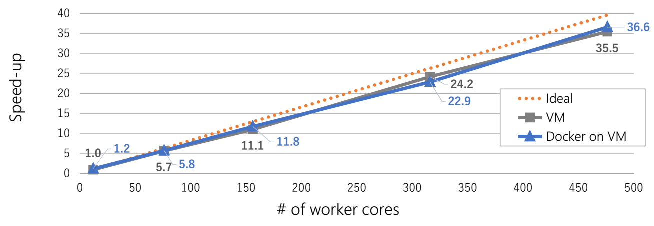

Fig. 6 Strong-scaling performance of MEGADOCK (based on VM=1) on the benchmark dataset

The execution time of MEGADOCK running with the Docker containers on the VM instances and MEGADOCK directly running on the VM instances (MEGADOCK-Azure) is shown in **Fig. 5**. Each bar shows the execution time on the number of VM instances, and the error-bars show the standard deviation in the measurements. **Fig. 6** shows the scalability in strong-scaling for the same results. The label "ideal" indicates the ideal linear scaling to the number of worker cores.

According to the result of scalability, both of them achieved a good speed-up of up to 476 worker cores. It was ×35.5 speed-up in the case of directly running on the VMs, and ×29.5 in the case of the Docker containers on the VMs. The speed-ups were almost equivalent in this experiment and they indicate that the performance overhead on the Docker containers is small when running on the VM instance of the cloud environment.

The MEGADOCK execution load is mainly composed of independent 3D-fast Fourier transform (FFT) convolutions on each single node even in the MPI version such that it tends to be a compute-intensive workload, not a data I/O or network-intensive; therefore, similar to the situation mentioned in the Linux container performance profile reports [7], MEGADOCK calculation on the distributed containers environment also performs well.

5. Discussion

In the experiment, the execution on a single VM instance with the MEGADOCK-Azure (VM) particularly consumed time but the reason was unknown, and that affected the result of scalability. This irregularity increases the scalability more than expected. The reason should be investigated; however, it is difficult because it is time-consuming calculation.

We did not achieve multiple GPU nodes parallelization using the MPI library in this study; however, now the VM instances attached to NVIDIA GPU devices are generally available. Moreover, there are more sophisticated VM instances for the HPC applications on Microsoft Azure that is connected by the InfiniBand each other and supports low-latency communication using remote direct memory access (RDMA). We have already performed experiments over multiple VM instances with GPUs [14] or HPC instances; therefore, further experiments with the Docker containers is our future challenge.

6. Conclusion

We implemented a protein–protein interaction prediction system, MEGADOCK, using Docker containers and its networking functions on the VM instances of Microsoft Azure. We confirmed that the performance is almost equivalent to the same calculation directly performed on the VM instances through the benchmark experiment of protein docking calculations. Both when MEGADOCK directly runs on the VM and when it runs with the Docker containers of MEGADOCK on the VM, the execution speed achieved was almost equal even when the number of worker cores increased up to approximately 500 cores.

The containers enable us to isolate software dependencies and the system software stacks, which offer a great advantage to the users in sharing software packages through platforms, thereby making it easy to distribute the latest research achievement. Virtualization technologies have been evolved in the context of the cloud computing; however, in the current era, many research institutions have introduced several container environments into their HPC infrastructures. To improve productivity and retain scientific reproducibility, it is necessary to introduce such software engineering techniques into research activities.

Acknowledgment

This work was partially supported by KAKENHI (Grant No. 17H01814 and 18K18149) from the Japan Society for the Promotion of Science (JSPS), the Program for Building Regional Innovation Ecosystems "Program to Industrialize an Innovative Middle Molecule Drug Discovery Flow through Fusion of Computational Drug Design and Chemical Synthesis Technology" from the Japanese Ministry of Education, Culture, Sports, Science and Technology (MEXT), the Research Complex Program "Wellbeing Research Campus: Creating new values through technological and social innovation" from JST, Microsoft Business Investment Funding from Microsoft Corp., and Leave a Nest Co., Ltd.

References

[1] "Docker." https://www.docker.com/, [Accessed May 8, 2019].
[2] G. Stphane, "LXC - Linux containers." https://linuxcontainers.org/, [Accessed May 8, 2019].
[3] P. Di Tommaso, E. Palumbo, M. Chatzou, *et al.* "The impact of Docker containers on the performance of genomic pipelines," *PeerJ*, vol. 3, no. e1273, 2015.
[4] A. Paolo, D. Tommaso, A. B. Ramirez, *et al.* "Benchmark Report: Univa Grid Engine, Nextflow, and Docker for running Genomic Analysis Workflows," *Univa White Paper*, 2016.
[5] E. W. Biederman, "Multiple Instances of the Global Linux Namespaces," In *Proc. the 2006 Ottawa Linux Symp.*, vol. 1, pp. 101–112, 2006.
[6] "Docker Hub." https://hub.docker.com/, [Accessed May 8, 2019].
[7] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, "An updated performance comparison of virtual machines and Linux containers," In *Proc. IEEE ISPASS 2015*, pp. 171–172, 2015.
[8] L. M. Vaquero, L. Rodero-Merino, R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41(1), pp. 45–52, 2011.
[9] G. M. Kurtzer, V. Sochat, M. W. Bauer, "Singularity: Scientific containers for mobility of compute," *PLoS One*, vol. 12(5), pp. 1–20, 2017.
[10] "TOP500 Lists," https://www.top500.org/, [Accessed May 8, 2019].
[11] M. Ohue, T. Shimoda, S. Suzuki, *et al.* "MEGADOCK 4.0: An ultra-high-performance protein-protein docking software for heterogeneous supercomputers," *Bioinformatics*, vol. 30(22), pp. 3281–3283, 2014.
[12] M. Ohue, Y. Matsuzaki, N. Uchikoga, *et al.* "MEGADOCK: An all-to-all protein-protein interaction prediction system using tertiary structure data," *Protein Pept. Lett.*, vol. 21(8), pp. 766–778, 2014.
[13] "Microsoft Azure." https://azure.microsoft.com/, [Accessed May 8, 2019].
[14] M. Ohue, Y. Yamamoto, Y. Akiyama. "Parallel computing of protein-protein interaction prediction system MEGADOCK on Microsoft Azure," *IPSJ Tech. Rep.*, vol. 2017-BIO-49, no. 4, 2017.
[15] A. Kivity, U. Lublin, A. Liguori, *et al.* "kvm: the Linux virtual machine monitor," *Proc. the Linux Symp.*, vol. 1, pp. 225–230, 2007.
[16] A. Velte, T. Velte, "Microsoft Virtualization with Hyper-V," 2010.
[17] "Xen Project." https://www.xen.org, [Accessed May 8, 2019].
[18] "VMware - Virtualization Overview." https://www.vmware.com/pdf/virtualization.pdf, [Accessed May 8, 2019].
[19] "Moby project." https://mobyproject.org, [Accessed May 8, 2019].
[20] "Docker Machine." https://docs.docker.com/machine/, [Accessed May 8, 2019].
[21] T. Hayashi, Y. Matsuzaki, K. Yanagisawa, *et al.* "MEGADOCK-Web: an integrated database of high-throughput structure-based protein-protein interaction predictions," *BMC Bioinform.*, vol. 19(Suppl 4), no. 62, 2018.
[22] R. Chen, J. Mintseris, J. Janin, Z. Weng, "A protein-protein docking benchmark," *Proteins*, vol. 52(1), pp. 88–91, 2003.
[23] E. V. Wasmuth, C. D. Lima, "KEGG: new perspectives on genomes, pathways, diseases and drugs," *Nucl. Acids Res.*, vol. 45, pp. 1–15, 2016.