PROCEEDINGS OF SCIENCE

A switching mechanism of visualization middleware and application using Docker

Kazuya Ishida*
School of Engineering, Osaka University
E-mail: ishida.kazuya@ais.cmc.osaka-u.ac.jp

Yoshiyuki Kido
Cybermedia Center, Osaka University
E-mail: kido@ais.cmc.osaka-u.ac.jp

Susumu Date
Cybermedia Center, Osaka University
E-mail: date@ais.cmc.osaka-u.ac.jp

Shinji Shimojo
Cybermedia Center, Osaka University
E-mail: shimojo@ais.cmc.osaka-u.ac.jp

A Tiled Display Wall (TDW) is one of the visualization systems, which is appropriate for visualization of large-scale data. Although the scientific area needs an easier and cheaper TDW than the current one, a large-scale TDW is still too expensive to build individually. Given that, it is advisable to share the large-scale TDW among scientists who need to use it. To share a TDW, a variety of visualization middleware and applications should be installed on it because scientists use a different one of them depending on their data and purpose. However, there is a problem that it is difficult to coexist multiple visualization middleware and applications on a TDW because there are conflicts of library versions among them. To solve this problem, we propose a switching mechanism of visualization middleware and application using Docker. By using the proposed mechanism, administrators just need to deploy Docker images with visualization middleware and applications on a TDW, and then users can use them without the conflict problem. In the evaluation, the proposed mechanism is shown to have practically and scalability.

International Symposium on Grids and Clouds 2017 -ISGC 2017-
5-10 March 2017
Academia Sinica, Taipei, Taiwan

*Speaker.

PoS(ISGC2017)017

1. Introduction

The recent improvement of computer performance has made it possible to carry out large-scale simulations in various research areas. Generally, these large-scale simulations generate very large amount of raw data from intermediate and final results (hereinafter, such data is called "large-scale data"). For example, in the research area of chemistry, a single simulation of turbulent combustion processes produces terabytes of raw data [1]. For another example, in the research area of astronomy, simulations on the formation and evolution of large scale structures in the universe generates 50 terabytes of raw data [2]. These large-scale data are very difficult to understand intuitively for most scientists. Therefore, visualization is required to make large-scale data easily understandable.

To visualize data, scientists have to use a visualization system. A Tiled Display Wall (TDW) [3] is appropriate for visualization of large-scale data. A TDW can provide a virtual large screen by using multiple display monitors (shown in Figure 1). Scientists can put high-resolution images and movies obtained from large-scale data (hereinafter, these are called "visualization contents") on the virtual large screen without a lack of information. Furthermore, visualization contents on the virtual large screen can be observed by a lot of people simultaneously. Therefore, a TDW can provide not only a high-resolution and scalable visualization environment but also a collaborative and sharable research environment. This is why a TDW has been accepted in various research areas.

Figure 1: Tiled Display Wall (TDW)

On the other hand, a TDW is too expensive for most scientists to buy individually. In order to realize a large-scale screen, a TDW needs a high spec graphics processor unit (GPU) and a high throughput memory on the back end PC cluster system. The scientific area needs an easier and cheaper TDW than the current one. In view of the above-mentioned situation, we have been proposed a concept of a cloud service dedicated to visualization: VaaS (Visualization as a Service). VaaS is a type of cloud-based computing which provides the shared visualization system and the back end PC cluster system beyond the Internet. To realize VaaS in the world, we have been developed the flow control mechanisms and the system architecture for TDW middleware [4] [5].

However, there is a problem that it is difficult to coexist multiple visualization middleware and applications on a TDW. Visualization middleware and applications have various kinds of packages such as Scalable Amplified Group Environment (SAGE2) [6], ParaView [7] and COllaborative VIsualization and Simulation Environment (COVISE) [8], each of which depends on the different particular versions of system libraries and graphic libraries to use a hardware acceleration with

GPU. The conflicts of library versions among different visualization middleware and applications can cause trouble in the operation of a TDW.

In this paper, we propose a switching mechanism of visualization environments which is driven by using virtualization mechanism. Technically, we leverage Docker, an implementation of virtualization mechanism, to divide each visualization environment individually. The reminder of this paper is organized as follows. Section 2 describes the detail of the technical background and the problem of conflicts of library versions for shared use of a TDW. Section 3 briefly describes a proposed mechanism of visualization environments using Docker. Section 4 illustrates the evaluation of the proposed mechanism, in detail. We conclude our research and suggest future work in Section 5.

2. Technical background and problem

In this section, the technical background and the problem of this research are described. Moreover, we introduce some related work.

2.1 Tiled Display Wall (TDW)

A Tiled Display Wall (TDW) is one of the visualization systems, which can provide a virtual large screen by using multiple display monitors [3]. The architecture of a TDW is shown in Figure 2. The TDW is composed of two components: the multiple monitors and the PC cluster. The multiple monitors are arranged in a matrix on a wall and each of them is connected to any one node in the PC cluster. When the TDW creates a virtual large screen, as many processes of visualization middleware or applications as the number of monitors are launched on the nodes in the PC cluster. Visualization middleware and applications are the software which executes rendering processing on the PC cluster and sends the appropriate parts of the visualization contents to each monitor. There are a lot of kinds of visualization middleware and applications, each of which can respectively handle different data formats and perform different operations to visualization contents.

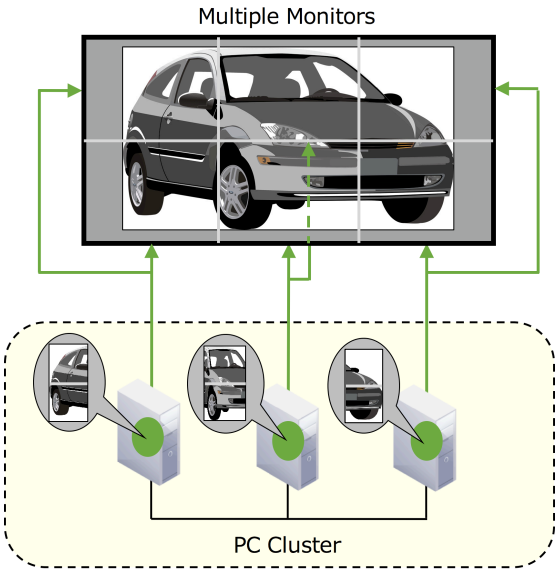

Figure 2: The architecture of TDW

2.2 Scalable Amplified Group Environment (SAGE2)

Scalable Amplified Group Environment (SAGE2) is an open-source visualization middleware developed by EVL (Electronic Visualization Laboratory) at the University of Illinois at Chicago [6]. SAGE2 has three major features. First, SAGE2 can display visualization contents obtained from data in remote sites by using Wide Area Network (WAN). Second, users can arrange multiple application windows on the virtual desktop provided by SAGE2. Third, SAGE2 allows application windows to be shared among TDWs in multiple locations.

The architecture of SAGE2 is shown in Figure 3. SAGE2 is composed of three components: the SAGE2 server, the display clients and the interaction client. In the architecture of SAGE2, multiple monitors are considered as the display clients, each of which has a different client ID. The SAGE2 server sends data streams of visualization contents to each display client by using WebSocket protocol. Each display client receives the data stream and display it by opening the URL corresponding to its client ID with a Chromium browser. Visualization contents on the display clients can be controlled by SAGE UI. To use SAGE UI, users have to open the URL for SAGE UI with a Chromium browser on the interaction client.

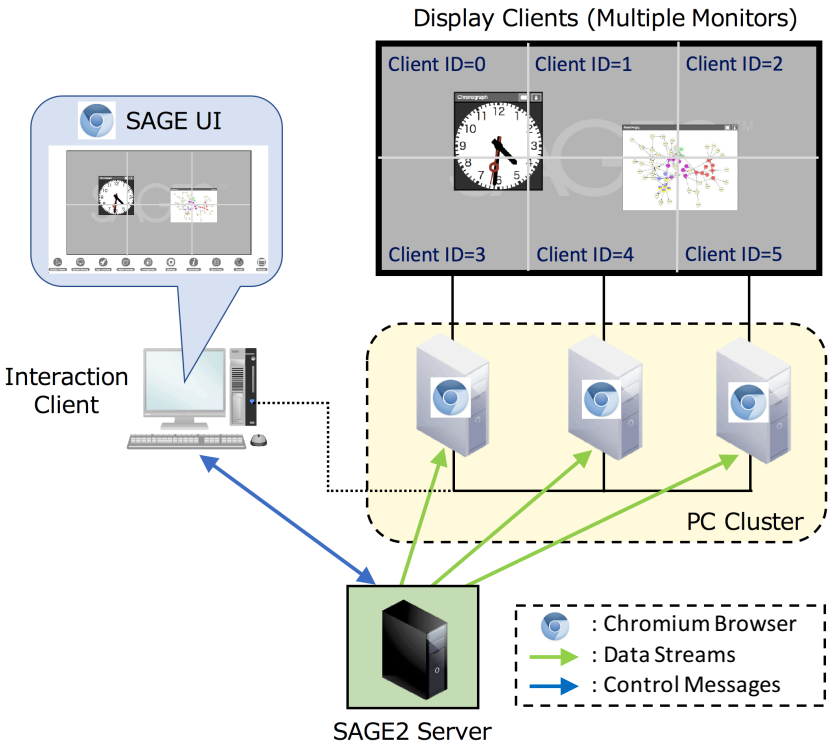

Figure 3: The architecture of SAGE2

2.3 ParaView

ParaView is an open-source visualization application developed by Kitware Inc. and Sandia National Laboratories and so on [7]. ParaView has two major features. First, ParaView can handle a variety of data formats such as VTK, OpenFOAM and Plot3D. Second, users can execute data processing to visualization contents flexibly and programmatically for data analysis.

The architecture of ParaView is shown in Figure 4. In the architecture of ParaView, each monitor has a different ID. On the PC cluster, the processes of ParaView (pvservers) are launched by using Messaging Passing Interface (MPI). The pvserver whose rank is N sends the appropriate part of visualization contents to the monitor whose ID is N by using X Window System. Visualization contents on the multiple monitors can be controlled by the GUI of ParaView on the client terminal, which is connected to the pvserver whose rank is zero.

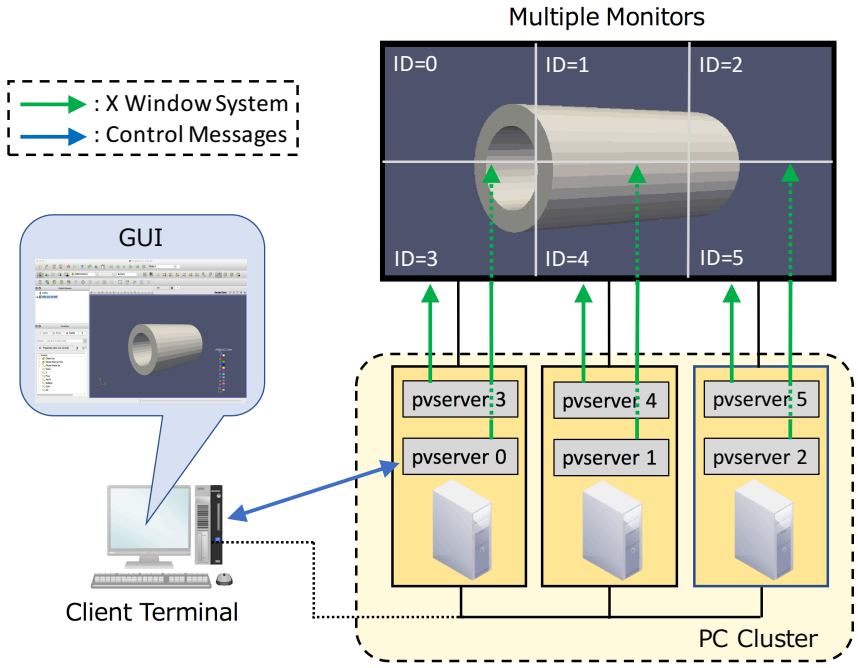

Figure 4: The architecture of ParaView

2.4 Difficulty in configuration

A TDW has a problem that configuration is very difficult for shared use because scientists need a variety of visualization middleware and applications on TDW such as SAGE2, ParaView and COVISE. Generally, visualization middleware and applications require the users to apply long and complicated configuration to the TDW environment. Furthermore, there are conflicts of library versions among different visualization middleware and applications because they depend on the particular version of system libraries and graphic libraries. These conflicts make configuration more complicated.

As an example, the libraries used by SAGE2, ParaView and COVISE are shown in Table 1. Table 1 suggests that there is a conflict of the version of FFmpeg between SAGE2 and ParaView. In addition, there is a conflict of the version of Python between ParaView and COVISE. When these visualization middleware and applications coexists on a TDW, these conflicts can cause trouble in the operation on the TDW.

2.5 Related Work

As with our research, there is some research for supporting multiple user environments on a shared cluster system by using Virtualization. National Energy Research Scientific Computing

SAGE2 (v2.0.0)	Node.js (v6.9 or later), ImageMagick (v6.9), FFmpeg (v3.0 or later)
ParaView (v5.1.2)	VTK (v6.0 or later), Python (v2.7), FFmpeg (v2.3)
COVISE (v2016.12)	VTK (v6.0 or later), Python (v3.0 or later), OpenSceneGraph (v3.2 or later)

Table 1: The libraries used by SAGE2, ParaView and COVISE

Center (NERSC) developed Shifter, which allows users to select and manage Docker images on a HPC cluster [9]. The feature of Shifter is the way to manage the images on a cluster. Instead of starting new containers on a cluster, Shifter extracts binaries and metadata from the Docker image and copies them to each node through a parallel filesystem.

Donggang *et al*. developed Docklet, which provides users with dynamic virtual clusters (Vclusters) on a shared physical cluster [10]. Users can run their HPC applications without any modification by creating, restarting and scaling in and out the Vcluster in browsers. To manage containers on a physical cluster, Docklet utilizes Linux Container (LXC) and OpenVSwitch instead of Docker.

Zhou *et al*. proposed DCSpark, which allows users to run multiple Spark applications on a cluster [11]. Users can concurrently run Spark applications whose configure settings and library dependencies are conflicted with each other on the same cluster. DCSpark differs from other frameworks in that DCSpark is designed to specialize in supporting Spark application environments.

As shown in above, most of the existing research focused on supporting multiple application environments on a HPC cluster. On the other hand, our research focused on supporting multiple visualization environments on a TDW. For example, our proposed mechanism utilizes Docker Swarm whereas other frameworks proposed in existing research often utilize the functions of a job scheduler. This is because the job schedulers are not used on the PC cluster of a TDW generally.

3. The proposed mechanism

In this paper, we propose a switching mechanism of visualization middleware and applications using Docker. The proposed mechanism enables users to switch and use the visualization environments on a TDW without the conflict problem. All administrators have to do is deploy Docker images with visualization middleware and applications on the TDW.

3.1 Docker

Docker is a recently emerging virtualization technology developed by Docker Inc. [12]. Docker can build and run Docker containers by using Docker images (the virtual images for Docker to package an application). The conflicts of library versions can be avoided by using Docker to separate the visualization environments from each other because the file system in a Docker container is isolated from the one of other containers. In fact, visualization environments can also be separated by using virtual machines (VMs). The reason why we used not VMs but Docker containers is because Docker has the following two advantages for realizing the proposed mechanism.

First, Docker containers are much lightweighter virtual environments than VMs. The difference between VMs and Docker containers are shown in Figure 5. VMs are the partitions managed by the hypervisor. VMs have to include an entire of the guest OS because they are completely

isolated from the host OS. On the other hand, Docker containers are the partitions managed by the Docker engine. Docker containers have only to include the application and its dependencies because they share the OS kernel with the host OS. This is why the applications in Docker containers can run with much less overhead than VMs. Furthermore, Docker containers can start and stop much faster than VMs because there is no need to boot and shutdown a guest OS in a Docker container.

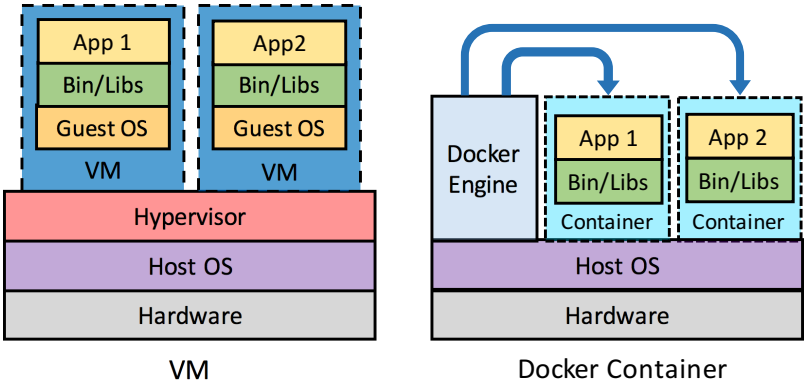

Figure 5: The difference between VMs and Docker containers

Second, Docker has the function named "Docker Swarm", which is suitable for implementing the switching functions of the proposed mechanism [13]. Docker Swarm can start up and stop all Docker containers on the multiple hosts simultaneously. Figure 6 illustrates the architecture of Docker Swarm. In the architecture of Docker Swarm, the hosts are unified into the Docker Swarm cluster. The Docker Swarm cluster is composed of the manager nodes and the worker nodes. The Docker engine on the manager node receives the Docker commands for Docker Swarm and sends the corresponding directions to control containers to all the nodes in the Docker Swarm cluster. In accordance with these directions, the Docker containers on the Docker Swarm cluster start, stop or reboot.

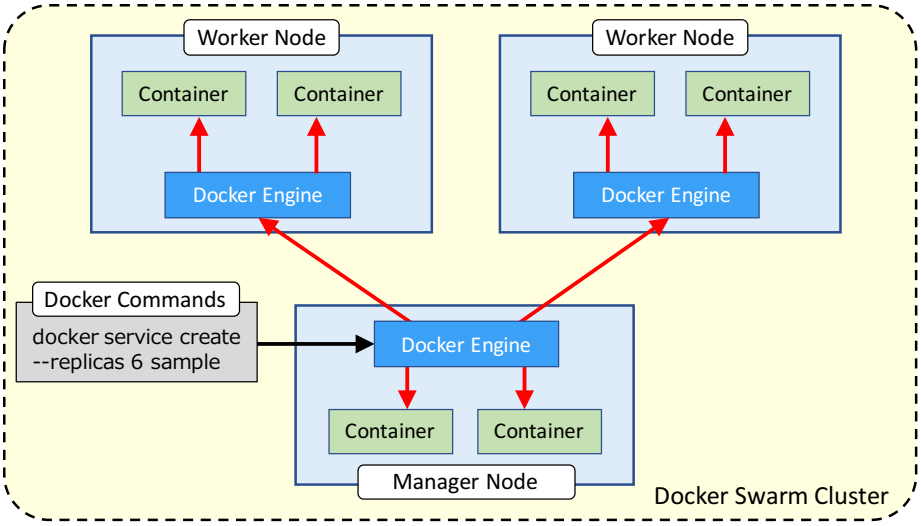

Figure 6: The architecture of Docker Swarm

3.2 The architecture of the proposed mechanism

The architecture of the proposed mechanism is shown in Figure 7. This mechanism is composed of three components: the web UI, the control server and the Docker Swarm cluster. The details of these components are explained as follows.

The web UI is the interface of the proposed mechanism, which is used on a web browser on a tablet PC. The web UI provides the buttons to switch visualization environments on the TDW. When one of the buttons is touched, the web UI sends a GET request with a parameter (a text which means the name of the touched button) to the control server.

The control server is the web server for the web UI, which is launched on the manager node in the Docker Swarm cluster. When the control server receives the GET request from the web UI, it executes a shell script for the switching corresponding to the parameter of this GET request. This shell script includes the Docker commands to start and stop containers on the Docker Swarm cluster.

The Docker Swarm cluster is built by unifying the nodes in the PC cluster. The Docker engine on the manager node gives directions to all the Docker engine in the Docker Swarm cluster in accordance with the Docker commands executed in the shell script. By following the directions, each node in the Docker Swarm cluster stops running containers and start new containers. In addition, initial settings for the visualization middleware or application are also performed in the new containers. In each container, only one process of visualization middleware or application is launched.

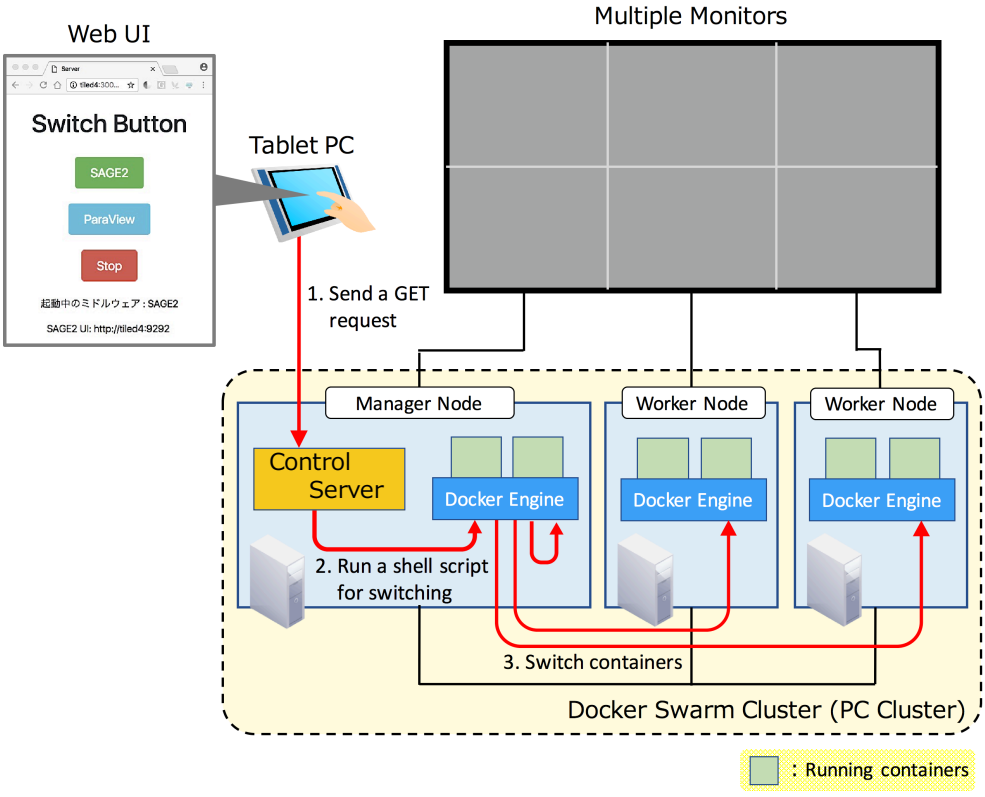

Figure 7: The architecture of the proposed mechanism

3.3 Implementation of the prototype

We implemented the prototype of the proposed mechanism, which has the functions to switch among three modes: SAGE2 mode, ParaView mode and Stop mode. SAGE2 mode is the state where SAGE2 is running on the TDW. ParaView mode is the state where ParaView is running on the TDW. Stop mode is the state where no visualization middleware or applications are running on the TDW. As described in the previous section, the switching is carried out by executing the shell script corresponding to the button touched on the web UI. The steps executed by each shell script are explained as follows.

(1) Switching to SAGE2 mode

1. If the TDW is in the ParaView mode, all the containers on the Docker Swarm cluster are stopped.
2. The container of the SAGE2 server is started on the manager node.
3. In the container of the SAGE2 server, the setting file of SAGE2 is modified and the SAGE2 server is launched.
4. The Chromium browser are started on each monitor in the fullscreen mode, opening the URL corresponding to its client ID.

(2) Switching to ParaView mode

1. If the TDW is in the SAGE2 mode, all the chromium browsers and the container of the SAGE2 server are stopped.
2. The containers of pvservers are started on the Docker Swarm cluster.
3. The hostfile for MPI is created by collecting the IP address of all the containers of pvservers.
4. The pvservers in the containers are launched by using MPI in accordance with the hostfile.

(3) Switching to Stop mode

1. If the TDW is in the SAGE2 mode, all the chromium browsers are stopped.
2. All the containers on the Docker Swarm cluster are stopped.

4. Evaluation

We report the evaluation results of the proposed switching mechanism. We have conducted two experiments in order to confirm the behavior of the proposed mechanism without a switching overhead. In the first experiment, we observed the switching time by using the proposed mechanism. In the second experiment, we measured the overhead of Docker in rendering processing.

4.1 Evaluation environment

The evaluation environment for the evaluation is shown in Figure 8. We constructed the 2 × 2 TDW for this evaluation. The PC cluster is used to build the Docker Swarm cluster which is composed of one manager node and three worker nodes. Each of the four nodes is connected to one monitor. The resolution of all the monitors is 1366 × 768 pixels. The web UI is used on the tablet PC. The specification of the four nodes in the PC cluster is described in Table 2. In addition, the specification of the tablet PC is indicated in Table 3. Finally, the versions of the software used in this evaluation are presented in Table 4.

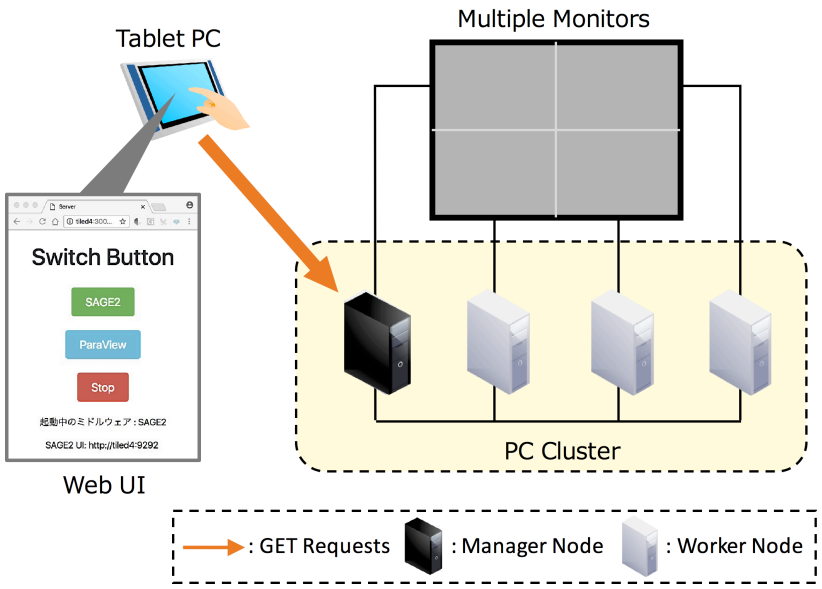

Figure 8: The evaluation environment

CPU	Intel Core i3-4150 (3.5GHz)
Memory	8.0GB
GPU	Intel HD Graphics 4400
OS	CentOS 7.3

Table 2: The specification of nodes in the PC cluster

CPU	Qualcomm Snapdragon 800 (2.15GHz)
Memory	2.0GB
OS	Android 4.4.2

Table 3: The specification of the tablet PC

Software	Version
Docker	1.12.6
SAGE2	2.0.0
ParaView	5.1.2
Chromium	55.0.2883.87

Table 4: The versions of software used in the evaluation

4.2 Evaluation method

4.2.1 Measurement of switching time

The prototype of the proposed mechanism has the following six switching patterns. In order to confirm that all the switching pattern can be completed within a short time, we observed each switching time with varying the number of processes of SAGE2 or ParaView. The number of these processes equals to the number of used monitors of the TDW.

(A) SAGE2 mode → ParaView mode
(B) SAGE2 mode → Stop mode
(C) ParaView mode → SAGE2 mode
(D) ParaView mode → Stop mode
(E) Stop mode → SAGE2 mode
(F) Stop mode → ParaView mode

The switching time is the sum of time for the following three phases. In order to confirm that which phase is dominant in the switching time, we also observed the breakdown of the switching time when the number of processes is four.

Phase 1: Stopping the remained containers on the Docker Swarm cluster

Phase 2: Starting the new containers on the Docker Swarm cluster

Phase 3: Launching the middleware or application in the new containers

4.2.2 Measurement of overhead of Docker

This measurement is carried out by applying the Python script shown in the below source code (test.py) with ParaView. If this script is executed, the time to rotate the visualization contents on ParaView by 360 degrees is measured. The visualization content used in this measurement is can.ex2, the sample 3D model of ParaView (shown in Figure 9). We measured the execution time of test.py with varying the number of pvservers and in case of both using and not using Docker and calculated the overhead of Docker according to the following formula.

$$(\text{The overhead of Docker}) = \left[\frac{(\text{The execution time of test.py in using Docker}\,[\text{s}])}{(\text{The execution time of test.py in not using Docker}\,[\text{s}])} - 1 \right] \times 100\ [\%]$$

```
# test.py

import paraview
import time

# Initialize a variable
average = 0

# Measurement is carried out 10 times.
for i in range(1, 11):
  # Get the start time
  start_time = time.time()

  # Execute the rotation by 1 degree 360 times
  camera = GetActiveCamera()
  for angle in range(1, 361):
    camera.Roll(1)
    Render()

  # Get the end time
  end_time = time.time()

  # Calculate the execution time
  interval = end_time - start_time
  average += interval

# Calculate the average of the execution time
average /= 10
print average
```

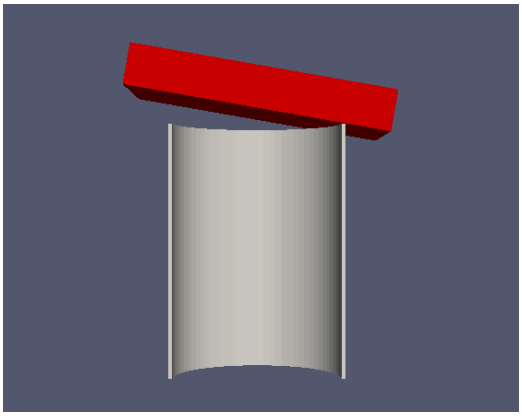

Figure 9: can.ex2

4.3 Evaluation result

4.3.1 Switching time

Figure 10 shows the result of the measurement of the switching time. The vertical line shows the switching time. The horizontal line shows the number of processes of SAGE2 and ParaView, which have to be started or stopped in switching visualization environments. Figure 10 suggests that the proposed mechanism can switch visualization environments on the TDW within 5〜20 seconds regardless of the number of processes. This is because the all containers on the PC cluster are started and stopped simultaneously thanks to Docker Swarm.

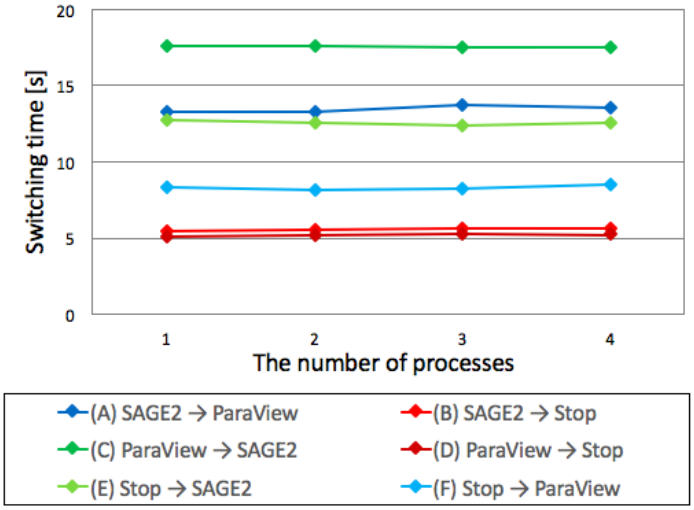

Figure 10: The switching time

Figure 11 shows the breakdown of the switching time when the number of processes is four. The vertical line shows the time for each phase. The horizontal line shows the executed switching pattern. Figure 11 suggests that the time for phase 3 is dominant in the switching to SAGE2 mode whereas the time for phase 2 is dominant in the switching to ParaView mode. This difference is caused by the preparation in the new containers. In the switching to SAGE2 mode, the shell script to generate SSL keys for the SAGE2 server is executed, which needs about six seconds to be completed. On the other hand, there is no need to do such a time-consuming preparation in the switching to ParaView mode.

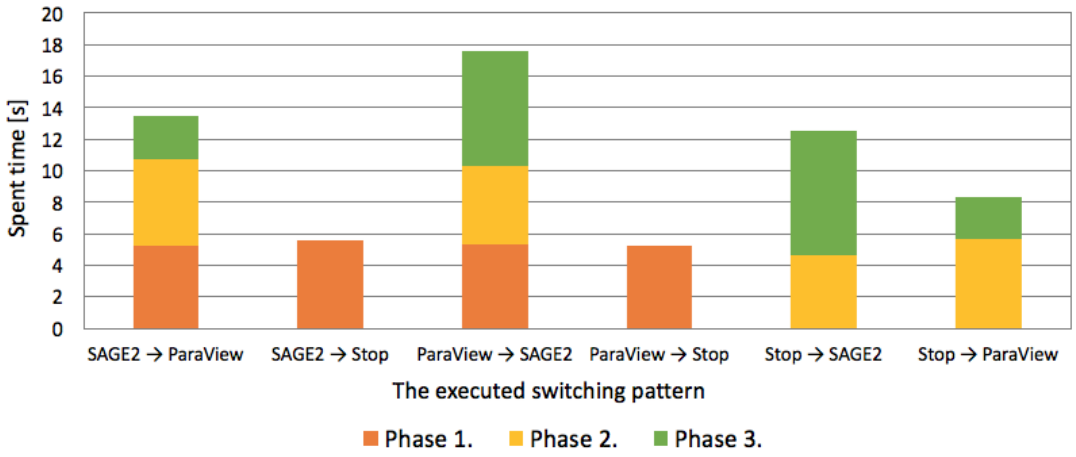

Figure 11: The breakdown of the switching time

4.3.2 Overhead of Docker

Figure 12 shows the result of the measurement of the overhead of Docker. The vertical line shows the percentage of the overhead. The horizontal line shows the number of pvservers. Figure 12 suggests that the overhead of Docker is kept about 3% as the number of pvservers in-

creases. This overhead is likely to be caused by the latency in the data transfer processing among the pvservers by using Docker network (the function of Docker to communicate among containers). The reason why the overhead is kept 3% regardless of the number of pvservers may be because ParaView can minimize the latency by using multiple buffering of OpenGL.

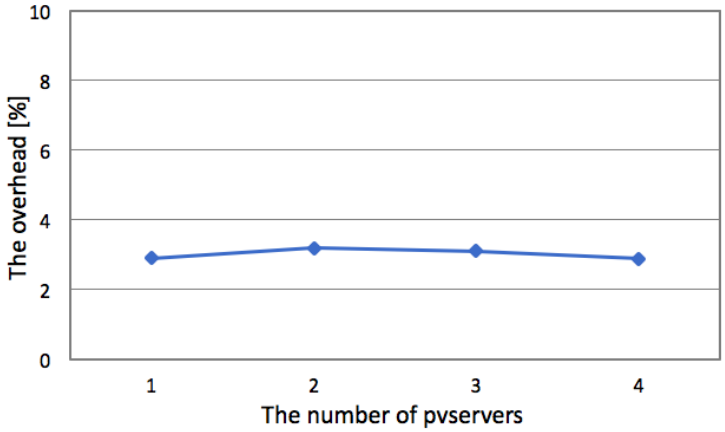

Figure 12: The overhead of Docker

5. Conclusion

In this research work, we have developed a switching mechanism of visualization middleware and application using Docker. By using the proposed mechanism, administrators just need to deploy Docker images with visualization middleware and applications on a TDW, and then users can use them without the conflict problem. To evaluate the proposed mechanism, we prototyped the proposed mechanism on a 2 × 2 TDW and conducted two experiments. First, we measured the switching time with varying number of processes of SAGE2 and ParaView. Second, we measured the overhead of Docker in rendering processes of ParaView with varying the number of pvservers. The results of these measurements suggest that the proposed mechanism has practically and scalability.

For the future work, we will apply the new functions for switching to other visualization middleware and applications such as COllaborative VIsualization and Simulation Environment (COVISE), The Cross Platform Cluster Graphics Library (CGLX) and Application Visualization System (AVS). Through this effort, we aim to realize a cloud service dedicated to visualization: Visualization as a Service (VaaS).

Acknowledgments

This work was partially supported by JSPS KAKENHI Grant Number 26540053.

References

[1] T. Oster, D. J. Lehmann, G. Fru, H. Theisel and D. Thévenin, *Sparse representation and visualization for direct numerical simulation of premixed combustion*, in *Eurographics Conference on Visualization*, vol. 33, Jun., 2014.

[2] K. Lee and J. Park, *A Hadoop-based output analyzer for large-scale simulation data*, in *Proceedings of the 2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, pp. 197–200, Dec., 2014.

[3] G. Humphreys, I. Buck, M. Eldridge and P. Hanrahan, *Distributed rendering for scalable displays*, in *the proceedings of the 2000 ACM/IEEE conference on SuperComputing*, no. 30, 2000.

[4] Y. Kido, K. Ichikawa, S. Date, Y. Watashiba, H. Abe, H. Yamanaka et al., *SAGE-based Tiled Display Wall enhanced with dynamic routing functionality triggered by user interaction*, *Future Generation Computer Systems* **56** (2016) 303–314.

[5] A. Endo, Y. Kido, S. Date, Y. Watashiba, K. Kiyokawa, H. Takemura et al., *Improvement of scalability in sharing Visualcasting contents for heterogeneous display environments*, in *the proceedings of International Symposium on Grids & Clouds 2016*, Mar., 2016.

[6] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot et al., *SAGE2: A new approach for data intensive collaboration using scalable resolution shared displays*, in *Proceedings of the 2014 IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 177–186, Oct., 2014.

[7] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens and J. Favre, *Remote large data visualization in the paraview framework*, in *Proceedings of the 2006 Eurographics Symposium on Parallel Graphics and Visualization*, pp. 163–170, May, 2006.

[8] A. Wierse, *Performance of the collaborative visualization environment (COVISE) visualization system under different conditions*, in *Proceedings of the 1995 International Society for Optics and Photonics's Symposium on Electronic Imaging: Science & Technology*, vol. 2410, pp. 218–229, Apr., 1995.

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

[10] W. Cui, H. Zhan, B. Li, H. Wang and D. Cao, *Cluster as a Service: A container based cluster sharing approach with multi-user support*, in *Proceedings of the 2016 IEEE Symposium on Service-Oriented System Engineering*, pp. 111–118, March, 2016.

[11] Z. Lei, H. Du, S. Chen, C. Zhu and X. Liu, *DCSPARK: Virtualizing spark using docker containers*, in *Proceedings of the 2016 International Conference on Audio, Language and Image Processing*, pp. 13–18, July, 2016.

[12] C. Kan, *DoCloud: An elastic cloud platform for web applications based on Docker*, in *Proceedings of the 2016 International Conference on Advanced Communication Technology*, pp. 478–483, Jan., 2016.

[13] N. Naik, *Building a virtual system of systems using docker swarm in multiple clouds*, in *Proceedings of the 2016 IEEE International Symposium on Systems Engineering*, pp. 1–3, Oct., 2016.