Research on Large Screen Visualization Based on Docker

Xin Liu*, Wenfeng Shen, Baohua Liu, Qin Li, Rong Deng and Xuehai Ding

Shanghai Institute for Advanced Communication and Data Science, School of Computer Engineering and Science, Shanghai University, Shanghai, China

*E-mail: liuxin2013@shu.edu.cn

Abstract. Docker is a system-level virtualization container technology, which is commonly used to package application software and provide rapid deployment services based on non-graphical interfaces to resolve dependency conflicts and version iteration problems. However, there are applications that require graphical user interfaces and user interactions, such as large-screen visualization. This work proposes a solution to use Docker and GPU to deploy large-screen visualization applications to display images, videos, WebGL applications, etc. We built a Docker cluster and deployed a large-screen visualization system by launching a microservice, which contains 28 containers, and mapping the host's X11 (X11 is widely used to provide a graphical interface to Unix/Linux users) Unix socket. At the same time, we tested the performance and compared it with the circumstance that applications ran on local host. It is concluded that the solution can achieve a good result when applied to large-screen visualizations with acceptable performance degradation.

1. Introduction

The current container technology such as Docker is developing rapidly in the field of cloud computing, and a new position DevOps has emerged. Nowadays, many cloud-computing companies begin to lay out on container cloud. The initial purpose of container technology is to quickly deploy software and solve the problem of environmental dependency conflicts. Container technology is in a way similar to traditional virtualization technology, but is also somewhat different. The general idea of traditional virtualization technology is to divide physical server's memory and CPU into virtual machines through virtualization technology. Each virtual machine has its own independent operating system and kernel. This virtualization technology is hardware-level virtualization. The container is a system-level virtualization technology. All containers on each host share the host's kernel. The isolation between containers is based on process-level isolation. Linux containers can be an alternative to traditional virtualization technologies because of its high resource utilization and less overhead [1]. The difference between a virtual machine and a container can be seen very clearly in Figure 1 and Figure 2.

![Figure 1. Traditional virtualization.](image1)

![Figure 2. System-level virtualization.](image2)
The components needed by the application are all packaged into a single image which can be reused. The comparison between the traditional software deployment model and the Docker technology-based deployment model is shown in Figure 3. It can be seen from the figure that the biggest difference between them is that each application of the traditional deployment mode shares the same library, and each application program based on the Docker deployment mode has its own independent library. This solves the problem of environmental dependency conflicts. For example, app1 requires python2.7, and app2 requires python3.0. If traditional deployment solutions are used to deploy two applications on a single server, it will cause environmental conflicts. It may be possible to solve this problem by purchasing a new server, but it will need an extra cost. Obviously, the use of Docker technology can be a very good solution for this problem.

![Figure 3. Container vs traditional deployment model.](image)

Secondly, the Docker image is decoupled from the operating system. Therefore, the same image can be run on multiple different platforms. For example, a same tomcat image can be run on a Centos environment or on an Ubuntu environment. This cross-platform feature is more often used for multiple environment software deployments in the enterprise.

Finally, the container is also used for software version control. According to different versions of the software packaged into different images, you can quickly upgrade and roll back the production environment.

Docker is also used in other fields including, big data processing [2], biology, astronomy [3] and materials science. Many people began to use Docker technology to apply scientific research to speed up the construction of test and computing environments, which could save them a lot of time. Container technology is mainly used for background running of the program without a graphical user interface (GUI). However, there are many applications that require GUI support to interact with user. Docker officially does not provide such functionality. In order to allow Docker to run GUI applications, some works proposed to install VNC(virtual network computing is a type of remote-control software that makes it possible to control another computer over a network connection) services inside the running container. Users can access the GUI program in the container through the VNC client or a browser. Some works also proposed to share the host's X11 service to solve GUI program display problems. Scientists in the related field of material gene have begun to apply these two methods to the application of GUI programs in the researches of molecular structure of materials, genes and so on. However, these methods are limited to some relatively simple graphic displays. For those more complicated applications such as WebGL, the performance is always not very good. In addition, Docker's application research on large-screen visualization applications such as sage2 still remains vacant.
2. Related work
Our work is based on several previously important work and existing technologies. Microservice architecture is now adopted by more and more companies. Using Docker to implement microservice architecture is the mainstream choice today [4]. It can greatly improve work efficiency.

In addition to using the host's regular resources such as CPU, memory and storage, Docker can also use host’s devices such as GPUs, FPGAs, and MICs in HPC [5]. Docker provides the approach to mount the host devices when running an image. In this paper we use GPUs to accelerate GUI display in containers with the nvidia-docker plugin. NVIDIA open the plugin’s source code in GitHub(https://github.com/NVIDIA/nvidia-docker).

Docker technology is also applied to machine learning [6]. In recent years, machine learning has been successfully applied in various fields. A number of machine learning software frameworks have also been introduced, and these software frameworks have been frequently updated in order to accommodate new hardware functions and software requirements, so that users have to frequently install new versions of the software, which is cumbersome. We can use container technology in field of machine learning, the environment software of various machine learning is packaged into different images according to different versions. The software provider updates the software, and at the same time, the public images are pushed by them. The images pulled by the user, so that the user can upgrade and run the new version of the software easily and quickly. The authors of that paper systematically evaluate the impact of Docker container on the performance of deep learning applications. They benchmark ed the performance of system components (IO, CPU and GPU) in a Docker container and the host system and concluded that running machine learning images with Docker will not cause noticeable drawbacks. Packaging a machine learning tool into a container is a viable solution.

GUIDock-X11 [7] uses Docker technology to share the X11 service of the host to implement biological software running with GUI to analyse RNA sequences. Running Docker image with some command parameters to mount the host’s X11 file and the container will transmit the data that needs to be displayed to the X11 service of the host for display.

GUIDock-VNC [8], the VNC service is installed inside the container, and the GUI program running in the container will output the data which is needed to display through the VNC, and the user can authorize access through the web browser.

Sage2 [9] is a new approach for data intensive collaboration using scalable resolution shared displays. We can use this way to connect multi block LED displays with server clusters, and splice it into super high resolution large screen for data visualization analysis.

3. Our work
We proposed deploying large-screen visualization applications using Docker with GPU. In this work, we use 8 workstations to create a large screen with a resolution of 13440x4320 by deploying the open source multi-screen sage2 deployment program with Docker. The configuration of the eight workstations is the same. The detailed configuration of the device is shown in Table 1.

| Item    | Config                                      |
|---------|---------------------------------------------|
| OS      | Ubuntu16.04                                 |
| Memory  | 64GB                                        |
| Network | SFP-10G                                      |
| Disk    | SATA 2*1T                                   |
| CPU     | Intel(R) Xeon(R) CPU E5-2623 v4 @ 2.60GHz 8 Core |
| GPU     | Nvidia Quadro M5000 8GB                     |
Among the eight workstations, we used seven workstations as display nodes and one workstation as a control node. The detail topology is as Figure 4. We used 28 LED display screens to form a maximum resolution of 13440x4320. Docker 17.03 is installed on each node, and these 7 display node servers form a container cluster. Each node starts 4 containers to output the display data to screen. In order to use the GPU application inside the container, we installed the nvidia-docker plugin on each node. Now we discuss how deploy our large-screen visualization application in two different ways.

3.1. Docker with GPU use local X11
In this way, we use Docker with GPU to deploy the large-screen visualization system. The topology is shown in Figure 4.

![Figure 4. Large screen system deployment diagram with Docker.](image)

We created a Docker image based on Ubuntu 16.04 and configured some sage2 configuration information. At the same time, we copied some NVIDIA's driver files to the inside of the container to support the WebGL application (we tried running the image directly through command “nvidia-docker run xxx” and found that WebGL does not work normally). Because this method uses local X11 server, we also set default environment variable ‘DISPLAY=:0’. Part of the docker-compose file content is shown in Figure 5. In this file we tell Docker cluster need to start running on each node 4 containers, e0, e1, e2 and e3. These four containers will mount the host's files ‘/state’ and ‘/tmp/.X11-unix’. The file ‘/state’ records the unique id of each node. A microservice application was formed and we managed our container based on the open source container management software Rancher(https://www.cnrancher.com/). In Figure 6 we can see that our current big screen display is running as a microservice and there are totally 28 containers.

![Figure 5. docker-compose.yml file.](image)

![Figure 6. sage2 client microservice.](image)

We set the “io.rancher.scheduler.global=true” attribute to tell Rancher that each node must run one container with this image. This means that when the health check component finds that one container in the cluster has stopped due to a program failure, Rancher will automatically start a new image on that node. In addition, we use Docker to run large-screen visualization software in the form of microservices, which can take the advantage of rapid version upgrades (Figure 7) and rollbacks.
(Figure 8) of microservices. As shown in Figure 7, if we need to upgrade our client we only need to modify the URL of the new image and click Upgrade.

After the sage2 microservice is started, we can see that the 28 large-screen spliced LED displays can play normal pictures, WebGL animations (Figure 9), videos and other applications.

3.2. Docker with GPU use remote X11
In this scenario, we don't need to install Docker service for each display node in Figure 10. The only thing we need to do is to configure the X11 service to allow it to be called remotely. The results of the test are somewhat regrettable. The actual test of this solution shows that only some picture applications can be played. Videos, WebGL and other applications are not supported, and the GPU cannot be used to accelerate the application.
3.3. Performance of deployed by Docker

We tested and recorded the average fps for different resolution videos and WebGL animations of different complexity by running the large screen visualization software sage2 client locally and running the sage2 client on Docker.

First we tested 720p, 1080p, and 2160p videos. We sampled 100 points of data, as shown in Figure 11 to Figure 13. It can be found that there are slight FPS drops running on Docker.

Then we tested the WebGL with an online demo(http://webglamples.org/aquarium/aquarium.html), which could modify the animation complexity simply by setting the amount of fish. The obtained FPS data is shown in Figure 14.

![Figure 11. Testing with 720P video](image1)

![Figure 12. Testing with 1080P video](image2)

![Figure 13. Testing with 2160P video](image3)

![Figure 14. Testing with WebGL demo](image4)

The above lines represent the average fps value.

From the comparison of the data in the above figures, it can be seen that the sage2 client running on the container does not have much performance difference with that running on the host.

4. Conclusion

The use of Docker to deploy large-screen visualization software systems can provide the following advantages:

- Take full advantage of resources. Display nodes can use their computing resources to do some other calculations when they are not used.
- Efficient cluster management. We can use third-party container cluster management software such as k8s, Rancher, etc.
Convenient version upgrade. As described in section 3.2, when sage2 has a new version, we can easily upgrade our services.

Rapid deployment. When a node fails, we can quickly restore the service.

In this paper, we proposed Docker's application on large-screen visualization software, and captured FPS data of video and WebGL applications on a large-screen visualization system based on Docker deployment. Compared with the local deployment model, we demonstrated the deployment on Docker. Large-screen visualization software can achieve a good result with the expense of a small performance impact. It shows that Docker is applicable to the field of large-screen visualization.

Acknowledgments
This work is partially supported by The National Key Research and Development Program of China (NO.2017YFB0701600), Science and Technology Committee of Shanghai Municipality under grant No. 16010500400.

Reference
[1] Joy, A.M. (2015). Performance comparison between Linux containers and virtual machines. 2015 International Conference on Advances in Computer Engineering and Applications, 342-346.
[2] Naik, N.A. (2017). Docker container-based big data processing system in multiple clouds for everyone. 2017 IEEE International Systems Engineering Symposium (ISSE), 1-7.
[3] Morris, D., Voutsinas, S., Hambly, N.C., & Mann, R.G. (2017). Use of Docker for deployment and testing of astronomy software. CoRR, abs/1707.03341.
[4] Jaramillo, D., Nguyen, D.V., & Smart, R. (2016). Leveraging microservices architecture by using Docker technology. SoutheastCon 2016, 1-5.
[5] Canon, S., & Jacobsen, D 2016 Cray User Group, Shifter: Containers for HPC.
[6] Xu, P., Shi, S., & Chu, X 2017 IEEE Computer Society, Performance Evaluation of Deep Learning Tools in Docker Containers. International Conference on Big Data Computing and Communications (pp.395-403).
[7] Ling-Hong, H., Daniel, K., Bong, L. S., & Yee, Y. K 2016 Plos One, 11(4), e0152686, Guidock: using docker containers with a common graphics user interface to address the reproducibility of research.
[8] Mittal, V., Hung, L. H., Keswani, J., Kristiyanto, D., Lee, S. B., & Yeung, K. Y., 2017 Gigascience, 6(4), Guidock-vnc: using a graphical desktop sharing system to provide a browser-based interface for containerized software.
[9] Renambot, L., Marrinan, T., Aurisano, J., Nishimoto, A., Mateevitsi, V., & Bharadwaj, K., et al., 2016, Future Generation Computer Systems, ‘Sage2: a collaboration portal for scalable resolution displays’ 54, 296-305.