Live Migration of Docker Containers through Logging and Replay

Chenying Yu[1] and Fei Huan[2 *]

School of Information Security Engineering, Shanghai Jiao Tong University, China 200240

[1]fullmoon@sjtu.edu.cn, [2]huanfei@sjtu.edu.cn

Keywords: Live Migration, Docker, VM.

Abstract. The lightweight virtualization technique has made virtual machines more portable, work more efficient and easier to management. The time of live migration can be greatly reduce when applied to containers based on the lightweight virtualization technique, which is very beneficial to data centers. This paper presented a logging and replay approach for live migration of Docker containers. With this approach, both the down time and total migration time are reduced in the experiment. Compared with the approach for traditional virtual machines, the down time has reduced by 65%, 55% and 44% under three different scenarios, meanwhile the total migration time has reduced by 27%, 47% and 38% respectively

Introduction

Lightweight virtualization has attracted increasingly attention in recent years. Compared with traditional virtualization systems like KVM, Xen and VMWare, lightweight virtualization systems such as OpenVZ, LXC, VServer and Zones are more portable, efficient and easier to management. For the traditional virtualization systems, a complete virtual machine is required to be created, including the guest operating systems and the required libraries supporting the operation of certain applications. Instead, the virtual machines, or containers, based on the lightweight virtualization technique share the same host operation system and its libraries [1]. Thus the containers will obviously become much smaller than the traditional virtual machines, and more machines can be deployed on one physical host at the same time.

On the other hand, live migration of virtual machines is an important topic for data centers. During the process of live migration, the services offered by the virtual machines should operate as usual. The best result is that users cannot feel the interruption of services during the migration [2]. This made it difficult for designing good migration solutions. Considering the distinct characters of containers based on the lightweight virtualization technique, the migration time of these containers can be greatly reduced, giving users a better experience of the services offered by the machines in data centers.

This paper will present a live migration solution for the Docker container, an open-source lightweight virtualization engine for Linux systems [3]. The migration solution is achieved by the logging and replay approach which logs the non-deterministic events of the source container during the migration and iteratively transfers the log records to be replayed on the target container. With the help of implementing this live migration solution to Docker containers, data centers could deploy their cloud architectures based on containers more convenient and fast.

This paper will first introduce several traditional migration solutions to lead to the design of this migration approach. After explaining the design of the logging and replay migration solution for Docher containers, the paper will give an experiment to test and compare the performance with the traditional live migration solutions. The result of the experiment will be measured according to two key metrics, namely down time and total migration time.

Related Work

The pre-copy [4] approach is an often mentioned solution for live migration, as well as the default live migration approach for Xen virtual systems [5]. This approach will first copy the all the memory pages from the source host to the target machine, and record the dirty memory pages during this

process. Then it will iteratively update the dirty pages of the previous round on the target machine until the size of dirty pages to be updated reduced to a certain threshold. It will finally stop the source machine, updated the last portion of dirty pages and resume the normal operation on the target machine. This method can greatly reduce the downtime for live migration. However, it may become useless for memory-intensive workloads. That is to say, when the dirty page producing rate is higher than the transfer and update rate, the iteration stage will become infinite. Especially, the iterative pattern used by this approach is of real worth for developing live migration strategies.

Later, many live migration solutions followed have been brought forward as improved versions of the pre-copy approach. Since the key point of pre-copy is transferring dirty pages which takes a great proportion of the migration efficiency, those improved solutions always focused on how to transfer less data during the iterative stage. Jin et al. [6] presented an approach that compressing the dirty pages before transferring them to the target host. They used the Characteristic-Based Compression algorithm to encode the data of dirty pages at the source machine and decode the data at the target machine. The algorithm is adaptive in terms of network bandwidth and CPU utilization. Another approach presented by Ma et al. [7] used a Bitmap data structure to store the most frequent changed memory pages to be transferred at last, because there can be some pages that are more frequent modified than the others. Besides, Mohan and Shine [8] proposed an approach to transfer the log files recording the modification of memory pages instead of the dirty pages. The log files are computed through Xor operations of the original page and the dirty page. With the compressed log files, the amount of data to be transferred between the source and target machines can also be reduced.

Except the improved versions of the pre-copy approach, the post-copy [9] was presented as a modification of the pre-copy method. Instead of transferring all dirty pages iteratively, post-copy only transfers dirty pages when necessary. That is to say, the dirty pages will not be transferred at first. As the target machine runs, it will trigger the page fault from time to time. And when a page fault is triggered, the target machine will then ask the source to transfer the required page to it.

In addition, the trace and replay [10] approach, however, has opened up a new train of thought for live migration. The approach uses a tool called ReVirt which is generally used for intrusion detection to record the events happened after the checkpoint. Then it will iteratively replay those events on the target machine to make the memory state of the two machines converge to the same. Based on this strategy, a logging and replay approach is designed and presented by this paper for Docker live migration.

Proposed Approach

The logging and replay approach for Docker live migration can be divided into three main stages, as illustrated in Fig. 1.

Stage 1. Copy the image file. During the first stage, the image file of the source container will be duplicated to be started as the target one. What calls for special attention is that there will be less data to be exported as an image file since Docker is built based on an incremental file system called AUFS [3]. To start a Docker container, a base image is required. But after starting the container, all the later changes will not be recorded to the base image. Instead, those modifications will be stored as extra layers above the base image layer. Thus for data centers, the base images can be stored in a shared storage device to reduce redundant duplications. Therefore, in this case, the base image is not required to be copied. Only the extra layer is required to be copied and can be used to start a new container based on the new image in the target machine. An export instruction can be used to export the extra layer to a .tar image file, and the target machine can use an import instruction to start a new container. During the previous process, there can be some changes happened on the source machine which are not recorded in the image file. These changes will be recorded into a log file and be replayed iteratively in the next stage.

Stage 2. Iterative Log and replay. In this stage, the log file from the previous stage will be transferred to the target container, and be replayed there. And during this process, some new events

can happen on the source container, which will also be recorded as log files to be replayed on the target container later. Iteratively, the log files will continually be transferred to the target container until the size of the log file decreases to a certain threshold.

Stage 3. Stop and resume. When the log file to be transferred has become small enough, the source container will be stopped. And there will be one last log file to be transferred to the target container. After replaying all the log files received, the target container will resume and take over the job of the source container.

Experiment and Performance Evaluation

The experiment was performed on two identical hosts, and each host had one Docker container running on it. The host was equipped with one Intel Xeon E5-2620 CPU and 8GB DRAM. Both of the Docker containers used the same base image of the Ubuntu 14.10 system. The shared 1 TB storage can be accessed via the NFS protocol by both hosts. Each host had a 1000 Gbit/s NIC to transfer data with the other host. Both containers were configured to use 1GB of RAM.

Considering different scenarios, the experiment tested three different kinds of workload on the containers:

1) Daily use. The container is left idle for possible daily use.

2) Static web application. The Apache 2.2.12 was used to measure the performance a web of static content. 100 simultaneous clients are set to download a 64KB file from the web continuously.

3) Dynamic web application. This test was performed by the SPECweb99 benchmark. It automatically generates different concurrent requests to the web server, including dynamic standard GET operation, dynamic random GET operation, dynamic POST operation and so forth.

Two metrics, down time and total migration time, were measured respectively for each kind of workload. The down time is calculated from the stopping the source container to resuming target one during the third migration stage, while the total migration time refers to the total time from the first migration stage to the last one. The migration was repeated ten times for each work load, and the results listed below (Fig. 1) are the average value of the ten trials.

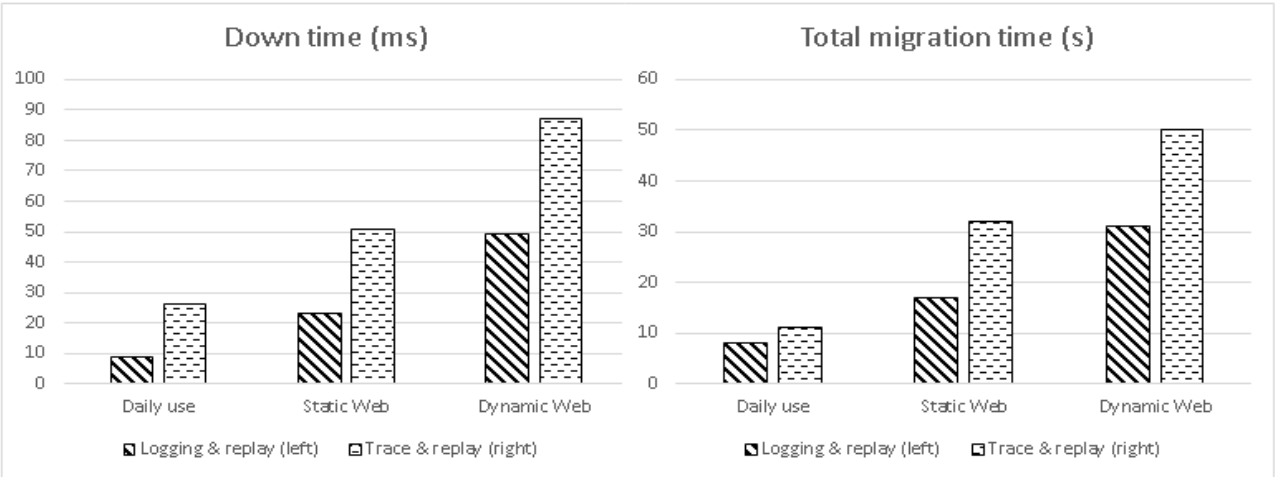

Figure 1. Down time and total migration time with different workloads and approaches

The experiment results are compared with the trace and replay approach [10]. Obviously, both the down time and total migration time are reduced when implementing the similar approach to Docker containers instead of traditional virtual machines. For down time, the results reduced by 65%, 55% and 44% for daily use, static web and dynamic web workloads respectively compared with the trace and replay approach, and the total migration time reduced by 27%, 47% and 38% with the three workloads respectively.

Summary

The containers based on the lightweight virtualization technique are more portable, efficient and easier to management. Both the down time and total migration time can be greatly reduced for live migration of Docker containers, helping improve the user experience when offering services from data centers. With logging and replaying iteratively, the Docker container is migrated from the source host to the target one. The down time reduced more significantly, from 44% to 65% under different scenarios, while the total migration time also reduced from 27% to 47% in different cases.

Acknowledgement

Deepest gratitude would like to be delivered to all those who have given advices and help on how to make this work possible. Especially the fellows in our lab, who have helped solve many problems during the research and design process of this work.

References

[1] N. Kratzke, A lightweight virtualization cluster reference architecture derived from open source paas platforms, Open Journal of Mobile Computing & Cloud Computing, 1.2 (2014) 17-30.

[2] L. Zhang, Scheme research of virtual machine live migration based on service live level, Computer Engineering & Applications, 49.19 (2013) 254-259.

[3] Information on https://www.docker.com/

[4] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt and A. Warfield, Live migration of virtual machines, USENIX Symposium on Networked Systems Design & Implementation, 2 (2005) 273-286.

[5] M. Sun and W. Ren, Improvement on dynamic migration technology of virtual machine based on Xen, International Forum on Strategic Technology, 2 (2013) 124-127.

[6] H. Jin, L. Deng, S. Wu, X. Shi and X. Pan, Live migration of virtual machines by adaptively compressing memory pages, Future Generation Comp. Syst., 38 (2014) 23-35.

[7] F. Ma, F. Liu and Z. Liu, Live virtual machine migration based on improved pre-copy approach, IEEE International Conference on Software Engineering and Service Sciences, IEEE Conference Publications, New York, 2010, pp. 230-233.

[8] A. Mohan and S. Shine, An optimized approach for live VM migration using log records, Computing, Fourth International Conference on Communications and Networking Technologies, IEEE Conference Publications, New York, 2013, pp. 1-4.

[9] M.R. Hines, U. Deshpande and K. Gopalan, Post-copy live migration of virtual machines, ACM Sigops Operating Systems Review, 43 (2009) 14-26.

[10] H. Liu, H. Jin, X. Liao, L. Hu and C. Yu, Live migration of virtual machine based on full system trace and replay, International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, 2009, pp. 101-110.