Enabling Container Live Migration Across Cloud Service Providers Using Page Server

Natawat Kwanpum
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
natawat.k@student.chula.ac.th

Kunwadee Sripanidkulchai
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
kunwadee@cp.eng.chula.ac.th

***Abstract*—Deploying containerized applications on the cloud computing platform is one of the most popular ways to deploy the application. Containerized application migration across the cloud providers is the freedom of users since they are not necessary to use the services of a single provider. However migrating container with large memory may take a long time. This project aims to develop the process of migration with a performance in a short time of migration, minimal downtime, the performance of application after migration by using page server to optimize overhead occurred by reading and writing files on disk several times during migration.**

***Index Terms*—containers, CRIU, live migration, cloud computing, page server**

I. Introduction

Container deployment is a method to build and deploy applications that are famous in current days. Unlike virtual machines, container abstract system in application level and shared kernel resources giving process to isolate CPU, memory, and network utilization managed with the container runtime. Moreover, containers also package all of the source codes of the application and its dependencies ensuring version compatibility between in several applications in one single machine and giving portability.

Running containerized applications on the cloud can happen in several options. First, use Infrastructure-as-a-Service (Iaas) model in the cloud providers e.g. AWS EC2 [1], GCP Compute engine [2] to run a server or virtual machine and run containerized applications inside it. This way users have to manage resources of servers and maintain servers by themselves. Second, use Platform-as-a-service (PaaS) model likes Amazon Elastic Container Service (AWS ECS) which is a fully managed container orchestration service providing security, reliability, and availability.

Anyway, each cloud providers offer different benefits e.g. pricing, ease of use, services. Container live migration allows cloud users to migrate their containers between any cloud providers to gain benefits or even migrate between two machines in the data center. Migration time depends on the memory used in containers. In real-world applications, we have a lot of traffic going in and out containers all the time and cause memory usage. Short migration time is one of the most important challenges in container live migration.

II. Background/Motivation

A. Live migration of containers

Instead of migrating the whole virtual machine, live migration is a method of detaching a group of processes which are running in term of the container, transfer to the other machine and attach them to run again. Container migration allows us to move containers around many machines to gain benefits e.g. load adjustment, resource management, and maintenance. Checkpoint/Restore In Userspace (CRIU) is a Linux software that freezes a running container (or application) and checkpoint memory state to disk. The memory state can be used to restore the container and run continuously from the checkpoint. The size of the memory state depends on the resource used by the container. In practice, live migration can take a long time to checkpoint, transfer, and restore. Improving the performance of live migration mainly focuses on minimizing downtime and total migration time. The downtime is the time that application cannot serve their functionality, say from stopping point of containers til finish restore containers at the destination. The total migration time covers from the start to the end of the migration process.

Many migration techniques have been proposed over the years. There are some variations of migration methods:

1) Normal migration. This migration method consisted of three simple steps. First, checkpoint by stop container and dump memory state to a disk. Second, transfer memory state files to the destination host. Finally, restore containers by using dumped memory state files.
2) Pre-copy migration. Instead of stopping container, this migration method copies all memory state, transfer memory to the destination (pre-copy) and keep containers running. Then, stop the container and track memory changes and migrate is as normal. So, this method can reduce the downtime of migration.
3) Post-copy migration. In the checkpoint step, this method copies a minimal state to the destination. After con-

tainers resume on the destination host, applications start reading faulted page memory from source host over the network.

In this paper, we focus on the pre-copy migration. Since we have seen the downtime of this method is quite low from an experiment and we tried to minimize downtime by using this method as a baseline.

B. Container migration support

Many container runtimes already support migration

Docker [3] is the most popular container platform to build images, push/pull from the repository, and run the containers. Docker is used to run containerized applications in the production environment of many companies and also supported by almost every PaaS in clouds. Docker provides command 'checkpoint' to save running container and start again. This command is experimental on the Docker daemon implemented by CRIU [4]. However, from our study, this command does not currently support features like pre-copy migration or page server.

Podman [5] is a daemonless container engine for developing, managing, and running OCI Containers. Podman is integrated with CRIU and support container live migration from version 1.4.0 (June 2019). Unfortunately, this container engine does not allow to do an advanced migration. Pre-copy, Post-copy and other options from CRIU are not compatible with Podman at this time. So we can implement the migration with page server with this container runtime. [6] [7]

RunC [8] is a tool for running containers according to the OCI specifications. RunC is an underlying technology of both Docker and Podman. The container running by RunC can be considered as a process. It allows us to fully manage and control and it works very well with CRIU. So RunC will be our container runtime of choice.

III. Related Work

Efficient Live Migration of Linux Containers [9] is the paper that presented several algorithms for container live migration that are currently available in CRIU. CRIU introduce new feature called "image cache/proxy". This feature allows for better total migration time and down time of the container applications that are migrated by avoiding the use of secondary storage. Unfortunately, the current state of image-cache/proxy is not compatible with the pre and post copy implementations of CRIU.

Voyager [10] is a container migration which related with our work. Container in Voyager uses union file system to support post-copy so downtime of this migration is just in time. In our paper, we focus on optimizing pre-copy method to achieve lower downtime and we achieve zero down-time with application-level solutions.

Enabling Live Migration of Containerized Applications Across Clouds [11] is a complete migration system that provide zero downtime, secure data transfer, Multi-cloud support, Interdependent container support, No Failed client connections, automated migration. This paper use pre-copy method and actually has downtime for under half a minute. Our goal is minimize this downtime and make migration better.

IV. System Design

In this section, we define the design goals and discuss detail about implementation.

A. Design Goals

1) Multi-cloud support. We desired to ensure containers can be migrated around any pairs of cloud providers
2) Short migration time. Main purpose of this paper, We tried to minimize the migration time
3) Secure Data Transfer. Since we transfer data between cloud over the internet, we want data to be transferred in a secure way.
4) Unavailability conceal. User will experience only delayed response during migration and not face unavailability of applications.
5) Automated migration: Live migration requires several steps to set up the environment on both source and destination machines. These steps should be automated and ease of use

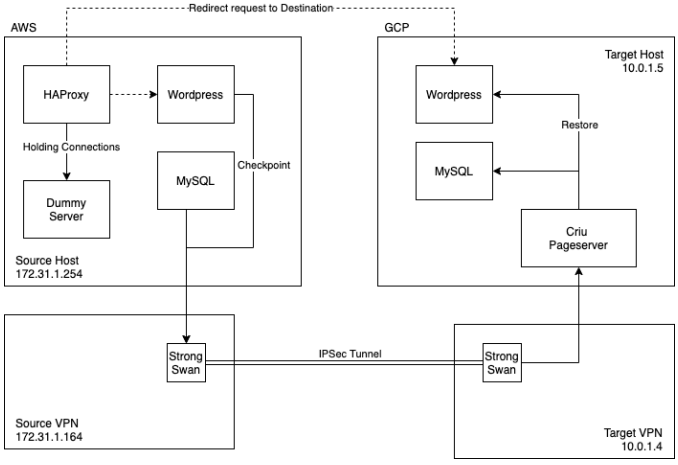

Fig. 1. Environment Setup

B. Multi-cloud support

Our system made of IaaS virtual machine to give an ability to fully manage in a Operating system(OS) level [12]. Although Docker is state-of-the-art container run-time in general software deployment, we use RunC which is underlying technology of docker to manage containers in this paper according to compatibility with CRIU

C. Short Migration Time

CRIU [4] provides multiple options to migrate process or container but we mainly focus on pre-dump method. We detected overhead on read and write memory page files on disk during checkpoint and transfer. So we decided to bring page-server to migration process.

Page server [13], as seen in figure 3, is a component of CRIU that allows to copy user memory to a destination host

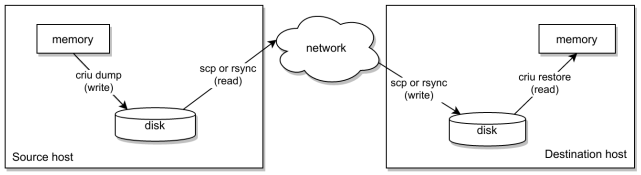

Fig. 2. Migration without page server.

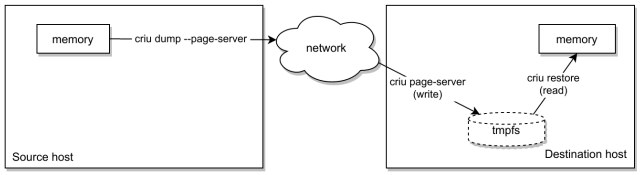

Fig. 3. Migration with page server.

rather than write to disk on a source host and transfer it by yourself. For any containers, the biggest part of transferring data is the memory used by the containers. Without the page server as in figure 2, migrating the memory in containers consist of these steps:

1) dumping the memory to files on disk
2) reading the files and sending over network to the destination host
3) receiving files on destination host and writing into disk
4) reading files from disk and restore to memory of container

All of the memory is written to disk and read from disk twice. It causes a significant I/O overhead and slows down the migration process. The overhead has more effect on the pre-dump method because of dumping memory more than one time.

When using page server memory is dumped not to disk but to network, allowing to eliminating read and write operation on a source host. On the other side, a destination host that running a page server is receiving data from the network and writing to disk.

D. Secure Data Transfer

Migrating containers between the clouds mean data will be transferred over the internet and this is not a good way to carry sensitive data. Virtual Private Network (VPN) protocols with IP security (IPSec) encryption is used in the system to be a protocol of communication between two communication points to provide data authentication, integrity, and confidentiality.

In setting up a VPN, most cloud providers also offer a VPN service in their cloud. However, we used VM installed StrongSwan as you seen in figure 1, placed in the same network of each host for implementing IPSec VPN to ensure that VPN instances work with any cloud providers.

E. Unavailability conceal

During migration users may experience unavailability of services in the downtime period, we implemented the dummy server with Flask. This dummy server in figure1 hold requests from users and make an interval health check on the target host to ensure that containers are ready to work again. After that, the dummy server returns response code 302 to perform redirection to the target host. Therefore users who are trying to connect to the applications during migration will face only just a delayed response, not an availability.

F. Automated migration

As we have known that set up VM and install packages and components in the system is quite complex, We used Terraform [14] to create, provision, and manage all of the infrastructures including hosts and VPN instances. To ensure that system has all of the required components and ready to proceed a migration, we used Ansible [15] to install packages and copy configuration files and scripts to both source and destination host.

In evaluation, we must do several repetitive experiments to get the accurate result, so we create Makefile to abstract the commands of executing Ansible playbook enabling to use our migration process easily.

V. Evaluation

A. Evaluation environment

To achieve multi-cloud support, we design to evaluate this method by migrating web server with database represent with Wordpress and MySQL between AWS and GCP

B. Methodology

The evaluation environment includes a source host on AWS and a target host on GCP. During migration, we simulate user traffic by Siege running on Azure virtual machine to create a workload that similar to 400 concurrent users on both containers.

VPN hosts are also in this environment to achieve secure data transfer between clouds. Specification of virtual machine used in this evaluation are shown in table I

TABLE I
Virtual Machines Specification

Host	Provider	Instance Type	vCPUs	RAM	Region
Source	AWS	t3.medium	2	4	ap-northeast-1
Source VPN	AWS	t3.small	2	2	ap-northeast-1
Target	GCP	n1-standard-1	1	3.75	asia-northeast-a
Target VPN	GCP	n1-standard-1	1	3.75	asia-northeast-a
Client	Azure	Standard D2s v3	2	8	Japan East

Evaluations were performed with scenarios of different workloads. Since the result of the evaluation is uncertain, we decided to evaluate each scenario ten times and measure with mean.

As we use Page Server to achieve shorter downtime during migration, we evaluate each migration method to compare the difference between them.

TABLE II
EVALUATION RESULT

Method	Pre-dump	Pre-copy	Pre-migration	Checkpoint	Transfer	Restore	Downtime	Total Time
Normal	-	-	-	9.44 ± 0.05	20.69 ± 0.85	6.32 ± 0.17	37.09 ± 0.79	37.09 ± 0.79
Page Server	-	-	-	18.62 ± 0.20	7.16 ± 1.07	6.01 ± 0.08	32.23 ± 1.11	32.23 ± 1.11
Pre-copy	10.79 ± 0.35	31.99 ± 10.7	42.79 ± 1.13	3.55 ± 0.10	10.77 ± 1.26	6.19 ± 0.16	20.98 ± 1.26	63.77 ± 1.58
PC & PS	21.01 ± 0.34	28.38± 1.37	49.39± 1.47	10.09± 1.87	4.35± 1.02	6.13 ± 0.12	21.15 ± 1.58	70.54 ± 1.83
Rsync	20.63 ± 0.47	28.96 ± 0.84	49.59 ± 1.11	11.24 ± 2.69	0.39 ± 0.01	6.05 ± 0.12	18.27± 2.69	67.87± 2.86
Rsync + HAProxy	18.22 ± 2.82	25.61 ± 3.93	43.83 ± 6.72	5.10 ± 1.89	0.33 ± 0.08	6.45 ± 1.03	12.29 ± 1.69	56.12 ± 7.45

[a] PC is refer to pre-copy and PS is refer to page server, [b] Rsync and HAProxy are optimizations on top of PC&PS method

C. Metrics

1) Pre-migration time and Migration time: Since short migration time if one of our goals, each step of migration is measure separately by python time library
2) Response time: During the migration process and evaluation, we used siege with a configure of 400 concurrent clients(400c) to add workloads. However, 400c can make the response time increase over time from 1 to 70 seconds due to resources of VM. So, we used Siege with 5c to generate workloads and measure the response time in the client perspective while this number of the concurrent client make web server can operate in a normal condition.

VI. IMPLICATIONS

First, we experiment to ensure that page server can reduce the downtime of migration, as you can see in table II page server method has 32.23 seconds of downtime beating normal migration which has 37.09 seconds of downtime.

When we are trying to use page-server with pre-copy method (PC&PS method in the table), in the first time, page server slightly increases downtime of migration from 20.98 to 21.13. From this point, we can see the transfer time of "PC&PS" method is 4.35 seconds. Rsync has overhead since it checks all the files in the entire container directory on both source and the target host. We decided to optimize it by checking only on checkpoint and pre-dump directory. Now when we look at method "PC&PS&Rsync" in the table, we can see that transfer time is nearly zero because we already transferred the most of files with page server and we achieve lower downtime at 18.27 seconds.

HAProxy [16] is used in our system to redirect the connection to containers, dummy server, and destination. During checkpoint in the migration process, we redirect all of the traffics from users to the dummy server to hold the connection and conceal unavailability of containers. Other benefits of HAproxy is also preventing containers from working on those request, from this point we can reduce checkpoint time from around 10.09-11.24 seconds to 5.1 seconds and can minimize downtime to 12.29 seconds.

From the response time perspective displayed in figure 4, the response time was increasing from around 0.5 seconds to around 1.3 seconds and may swing or spike on some requests.

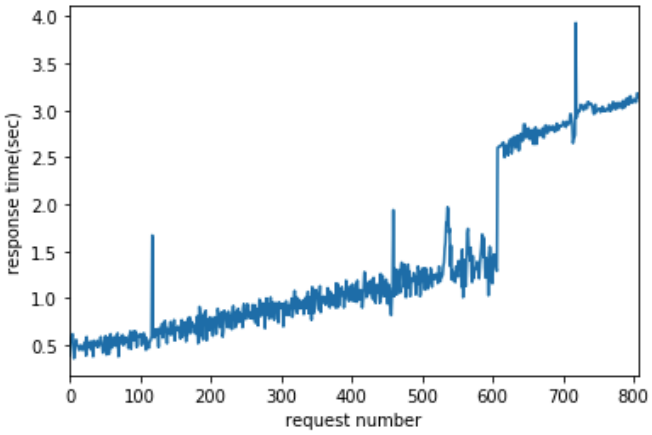

Fig. 4. Response time from client perspective.

Request number 600 showed that the migration process has done. After migration, response time is jump up to 2.5 and increasing due to redirection from HAproxy to target machine which has lower resources than source machine.(2CPUs vs 1CPU). However, it seems like clients do not face a significant downtime and web server is still available during migration from the client perspective.

Page server can help us to almost eliminate transfer time in migration which is overhead from I/O on the source and destination host and directly affect the downtime of migration. However, checkpoint time is increase when using page server due to sending memory state over the network to page server in checkpoint step but we can optimize checkpoint by preventing container using during checkpoint(hold the request for a while).

VII. CONCLUSIONS

In this paper, we presented container live migration using page server aiming to minimize downtime during migration. The evaluation results show that the page server can use to reduce read/write operation which is overhead in migration container with a large number of memory page files

REFERENCES

[1] D. J. Daly, "Amazon ec2," 1987. [Online]. Available: https://aws.amazon.com/ec2/

[2] "Compute engine: Virtual machines (vms) — google cloud." [Online]. Available: https://cloud.google.com/compute
[3] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014.
[4] "Criu." [Online]. Available: https://criu.org/
[5] "What is podman?" [Online]. Available: http://docs.podman.io/en/latest/
[6] "Podman." [Online]. Available: https://criu.org/Podman
[7] "Container migration with podman on rhel." [Online]. Available: https://www.redhat.com/en/blog/container-migration-podman-rhel
[8] Opencontainers, "opencontainers/runc," May 2020. [Online]. Available: https://github.com/opencontainers/runc
[9] R. Stoyanov and M. J. Kollingbaum, "Efficient live migration of linux containers," in *ISC Workshops*, 2018.
[10] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: Complete container state migration," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 2137–2142.
[11] T. Benjaponpitak, M. Karakate, and K. Sripanidkulchai, "Enabling live migration of containerized applications across clouds," Feb 2020. [Online]. Available: https://infocom2020.ieee-infocom.org/accepted-paper-list-main-conference
[12] J. Naren, S. Sowmya, and P. Deepika, "Layers of cloud – iaas, paas and saas: A survey," *International Journal of Computer Science and Information Technology*, vol. Vol. 5 (3), pp. 4477 – 4480, 06 2014.
[13] "Page server." [Online]. Available: https://criu.org/Page_server
[14] "Introduction to infrastructure as code with terraform." [Online]. Available: https://learn.hashicorp.com/terraform/getting-started/intro
[15] "Ansible documentation," May 2020. [Online]. Available: https://docs.ansible.com/ansible/latest/index.html
[16] J. E. C. D. L. Cruz and I. C. A. R. Goyzueta, "Design of a high availability system with haproxy and domain name service for web services," *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, 2017.

APPENDIX

Source code:https://github.com/NatawatK/live-migration.git