ISSN 2672-1562
VOL. 6

The 6th International Conference
on Next Generation Computing 2020

# ICNGC 2020

**Dates** December 17(THU) ~ 19(SAT), 2020

**Venue** Shilla Stay Haeundae, Busan, Korea

Organized by | 사단법인 한국차세대컴퓨팅학회 Korean Institute of Next Generation Computing

Sponsored by | IITP 정보통신기획평가원 TRACOM

## IV. CONCLUSION

This paper proposes a reference model for compatibility among the different SSI platforms by suggesting requirement of each SSI platform layer that is composed of a storage layer, resolution layer, and communication layer. We describe requirements of each layer and propose the reference model to satisfy the requirements. Then we explain how this reference model support compatibility among the heterogeneous SSI platforms

## ACKNOWLEDGEMENT

This work was supported by the Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No. 20HS3710, Development of Decentralized Self Sovereign Identity Management Technology using Blockchain).

## REFERENCES

[1] D. Reed, M. Sporny, D. Longley, C. Allen, R. Grant, M. Sabadello, and J. Holt, “Decentralized identifiers (DIDs) v1.0,” W3C, https://www.w3.org/TR/did-core, July 2020.
[2] M. Sporny, D. Longley, and D. Chadwick, “Verifiable credentials data model 1.0,” W3C, https://www.w3.org/TR/vc-data-model, November 2019.
[3] Sovrin, "Sovrin homepage," https://sovrin.org, August 2020.
[4] uPort, “uPort homepage,” https://www.uport.me, August 2020.
[5] Metadium, "Metadium homepage," https://www.metadium.com, November. 2020.
[6] Loopchain, “Loopchain homepage,” https://www.iconloop.com, November. 2020.
[7] Omnione, “Omnione homepage,” https://www.omnione.net/ko/service/omnioneservice, November 2020.
[8] DIF, “Decentralized-identity/universal-resolver,” https://github.com/decentralized-identity/universal-resolver.





# Timeline-Based Container Restore via a Computationally Efficient Snapshot

Sang-Hoon Choi
SysCore Lab.
Sejong University
Seoul, Korea
csh0052@gmail.com

Ki-Woong Park
Dept. of Information Security
Sejong University
Seoul, Korea
woongbak@sejong.ac.kr

***Abstract*—Recently, container technology has garnered attention for maximizing the utilization of computing resources. Container technology is widely used in micro-services, DevOps, etc. because it requires less resources than existing virtualization technology and can be operated in various environments. Therefore, stability is crucial in containers. Hence, when a container instance fails to provide service because of an external attack or its internal fault, a serious problem occurs. Investigation is necessitated for detection and restoration when a service error occurs inside a container. Herein, we propose a technique to efficiently capture a running container instance and restore it to the desired point in time.**

***Keywords—cloud computing; container; snapshot***

## I. INTRODUCTION

Recently, container technology has garnered attention in the field of cloud computing. Container technology is attracting attention because it can execute instances with fewer resources compared with virtualization technologies [1, 2]. Container technology can drive instances with low overhead because it shares the kernel of the host operating system. Because containers use most elements in conjunction with the host operating system, only a minimal number of resources is required to execute the instance. Recently, it has been widely used in serverless fields to utilize these features effectively

All services provided to the user are executed in the container instance. Therefore, stability is crucial in containers. For example, when a container instance fails to provide a service because of an external attack or its own fault, a significant problem arises. Herein, we propose a method to improve the safety of containers. Our proposed technology can efficiently capture and restore live container instances. The timeline snapshot framework records all information regarding instances executing in a container environment at specific intervals. When a problem occurs in the container service, the user can return to the point of time at any time. The system proposed herein provides three benefits. First, no modules need to be installed inside the container. All logging occurs at the host operating system level. Second, our system is computationally efficient. The memory workload test is reduced by approximately 37.8% compared with the full snapshot. Finally, we applied a container-specific restoration technique. A container-specific encoding scheme is applied in our system to remove unnecessary or duplicated data elements when restoring a container.

## II. RELATED WORK

Research has been actively conducted to efficiently store the state information of an instance in a virtualized environment [3, 4, 5]. The checkpoint on the docker uses a technique known as CRIU, instead of its own engine [6, 7, 8]. CRIU is a project to implement checkpoint/restore functionality for Linux. Most snapshot-related studies, including container migration, use CRIU. However, CRIU is not a technology specific to container instances. Therefore, the operation is also performed on data that overlap unnecessary data for container restore. Such an operation results in a significant amount of overhead when performing repetitive snapshots. Therefore, we propose a snapshot technique that can remove unnecessary data and redundant data in a container to solve the overhead problem.

## III. TIMELINE-BASED CONTAINER RESTORE FRAMEWORK

This section describes the framework designed to enhance the safety of containers. Our proposed framework can efficiently capture and restore live container instances.

### *A. Timeline-base Container Restore Framework*

Our goal is to maximize efficiency by removing unnecessary information in capturing the state information of container instances. Our system targets a container platform using a docker engine. We analyzed the factors required to drive a container on a docking engine. The docker container was driven by cgroups, namespace, and image (file system). Therefore, various information is required to dump/restore the state data of the container. However, a checkpoint utilizing a CRIU dumps three sets of unrequired data.

First, the checkpoint of the docker will dump hosts, hostname, resolv.conf, etc. from the host operating system. Because these files are used for restoring, they are required only at the point of restore. Therefore, it is unnecessary to dump data fixed to the host when obtaining a snapshot. Second, all mount files for the container instance are dumped. In a modern docker engine, container storage is managed using a file system known as Overlayfs [9]. Overlayfs allows one, typically read–write, directory tree to be overlaid onto another, read-only directory tree. All modifications are performed in the upper, writable layer. Utilizing these advantages when performing container snapshots maximizes efficiency. We only dumped the storage information in the upper directory when performing a container snapshot. Subsequently, we

managed the dumped data as a table such that duplicate data can be removed and dumped in the next snapshot. Finally, checkpoint performs a full memory dump on the container instance. Containers that perform typical operations have a large amount of duplicated memory data. We were cognizant of this problem and have performed studies regarding memory duplication data in virtualized environments [10]. We discovered that dumping the unchanged memory area is extremely inefficient. To solve the memory duplication problem, we designed a snapshot to initialize the dirty bit (memory) such that it can only be dumped to the changed memory.

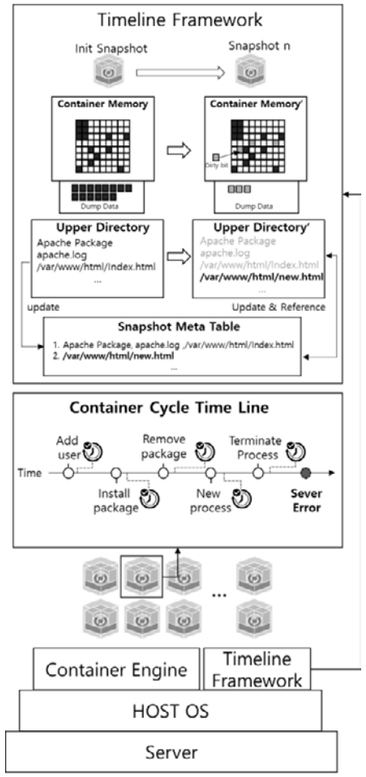


Fig. 1. Snapshot framework specialized for container timeline restore

The framework proposed herein is shown in Fig 1. We proposed three methods to maximize the efficiency of repetitive snapshots. First, we removed unnecessary host operating system files for container restoration. Files such as hosts, hostname, and resolv.conf are designed to be mapped when the container is restored. Second, we used Overlayfs to perform container snapshots. To devise a container-specific snapshot technique, we devised to dump only the modified data. Finally, the duplication memory area of the container was removed. It was designed to dump only the changed memory using the dirty bit. Because our snapshot technique is intended for timeline-based snapshot recovery, dependency exists between the snapshot data.

## IV. EXPERIMENTS

In this section, we verify the performance of our proposed system. Our performance verification process is as follows. First, Ubuntu 18.04 Image is executed in a docker environment. Second, a specific workload is created inside the container. Finally, we capture a snapshot of the container instance and compare the performance of the conventional container snapshot technology with our proposed scheme.

### A. Experiment Environments

We conducted the following experiments. We used Ubuntu 18.04 with 32 GB of memory and a normal HDD as a storage server and Ubuntu 18.04 64 bit as a container using a docker image. Furthermore, we installed Stress [15] inside the container to execute the stress test. Stress is a tool that imposes a configurable amount of CPU, memory, I/O, or disk stress on a POSIX-compliant operating system. We compared the data size variation of the container snapshot by memory and CPU workload. Finally, we performed a snapshot for 100 times in a cycle of 5,000ms.

### B. Performance Comparison by Workload

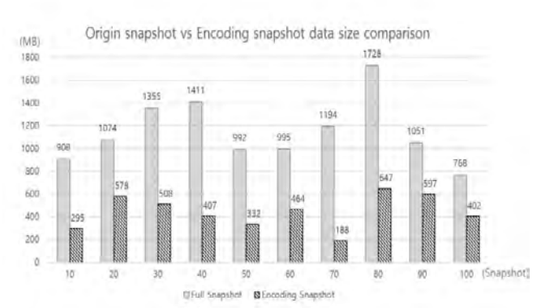


Fig. 2. Measurement of snapshot data change by memory workload Memory

To validate the performance of the container encoding snapshot module, we executed the workload process inside the container instance and performed 100 snapshots at 5000 ms intervals. The results for the memory workload experiment are shown in Fig 2. The memory stress option is "--vm 1." We compared the 10 accumulated snapshot data sizes. As a result of the experiment, 11.2 GB of data was created using full-snap and 4.3 GB of data was generated using an encoding snapshot. The encoding-snapshot size reduced to 37.8% compared with the full-snapshot size.

## V. CONCLUSION

Herein, we proposed a technique to efficiently capture and restore active container instances to enhance container safety. The benefits of the proposed technique are as follows. First, no modules need to be installed inside the container. All processes were executed on the host operating system. Second, the encoding snapshot was extremely efficient. The memory workload test reduced by approximately 37.8% compared with the full snapshot. Third, we applied snapshot technology specialized for containers. A container-specific encoding scheme was applied in our system to remove unnecessary or redundant elements when snapshots were captured. However, our proposed snapshot technology depends on previous dump data—a problem that we must be solved in the future.





## ACKNOWLEDGMENT

This work was supported by Institute for Information & Communications Technology Promotion (IITP) grants funded by the Korean government (MSIT) (No. 2018-0-00420) and supported by National Research Foundation of Korea (NRF) grants funded by the Korean government (NRF-2020R1A2C4002737).

## REFERENCES

[1] Morabito, Roberto, Jimmy Kjällman, and Miika Komu. "Hypervisors vs. lightweight virtualization: a performance comparison." Cloud Engineering (IC2E), 2015 IEEE International Conference on. IEEE, 2015.

[2] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." Linux Journal 2014.239 (2014): 2.

[3] Simha, Dilip Nijagal, Maohua Lu, and Tzi-cker Chiueh. "A scalable deduplication and garbage collection engine for incremental backup." Proceedings of the 6th International Systems and Storage Conference. ACM, 2013.

[4] Arcese, Mauro, Marco Mattia, and Stefano Sidoti. "Reconfiguring a snapshot of a virtual machine." U.S. Patent No. 9,817,685. 14 Nov. 2017.

[5] Mutalik, Madhav, and Srikanth Palaparthi. "Facilitating test failover using a thin provisioned virtual machine created from a snapshot." U.S. Patent No. 9,792,187. 17 Oct. 2017.

[6] Pickartz, Simon, et al. "Migrating LinuX containers using CRIU." International Conference on High Performance Computing. Springer, Cham, 2016.

[7] Nadgowda, Shripad, et al. "Voyager: complete container state migration." Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on. IEEE, 2017.

[8] Louati, Thouraya, et al. "Lxcloud-cr: towards linux containers distributed hash table based checkpoint-restart." Journal of Parallel and Distributed Computing 111 (2018): 187-205.

[9] Overlayfs, https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt

[10] Sang-Hoon Choi and Ki-Woong Park, 2017, "Computationally Efficient Instance Memory Monitoring Scheme for a Security-Enhanced Cloud Platform," Journal of the Korea Institute of Information Security & Cryptology, Vol. 27, No. 4, pp. 775~783.

[11] Geek, Hectic. "Stress Test Your Ubuntu Computer with 'Stress."