Reducing the Time of Live Container Migration in a Workflow

Zhanyuan Di, En Shao, Mujun He

Reducing the time of live container migration in a workflow

Zhanyuan Di[1,2][0000−0003−2716−5051], En Shao[1,2][0000−0002−9678−7228] ⋆, and Mujun He[3][0000−0003−3051−4304]

[1] University of Chinese Academy of Sciences, China, Beijing
[2] State Key Laboratory of Computer Architecture, Institute of Computing Technology, CAS, China, Beijing
{dizhanyuan,shaoen}@ncic.ac.cn
[3] Chongqing University, China, Chongqing
tbabm.he@gmail.com

Abstract. As a lightweight virtualization solution, container technology can provide resource limiting capabilities and can run multiple isolated process sets under a single kernel instance. Multi-tenant preemption leads to competition in computing, storage, and network resources, resulting in degraded computing service performance. Virtualization service migration can provide a solution to the problem of resource shortage in supercomputing systems. However, the resource overhead and delay in the migration process also reduce the efficiency of high-performance computers. To solve this problem, this dissertation proposes a method of container migration and a tool for supporting container migration. Also, this paper optimizes the startup of containers from checkpoints and proposes a multi-container migration strategy, reducing migration time by 30% compared to sequential migration. The migration method in this paper provides valuable experience for service migration in supercomputers and data centers.

Keywords: container · live migration · CRIU · Docker · workflow.

1 Introduction

With the popularity of Linux in the field of servers, more and more manufacturers provide users with Linux-based services. Operating system virtualization technology has also developed, which is lightweight virtualization technology. As a representative of virtualization technology at the operating system level, docker [6] has become more and more popular in recent years. Container virtualization technology is more like a lightweight alternative to the hypervisor, with higher resource usage efficiency than virtual machines. With the development of clusters, grids, and cloud computing, the application of virtualization technology is more widespread. The wide applicability of virtualization technology makes it a commonly used technology in cloud computing centers and has become the

⋆ The corresponding author is En Shao:(Email: shaoen@ncic.ac.cn)

basic architecture of cloud computing technology. It has also been applied in a large number of cloud-based supercomputers.

In the field of high-performance computing, service migration [20] based on virtualization provides an effective and feasible solution to solve the resource shortage of supercomputing systems. In the field of supercomputing system structure design, high-performance computers regard computing performance as their core. The computing resource overhead of service migration, the time overhead of service interruption, and the network overhead of data migration will all seriously affect the usage efficiency of high-performance computers.

Virtualization service migration is divided into cold migration and live migration [12]. The main discussion in this paper is live migration technology. Compared with cold migration, live migration saves the entire service running state and directly transfers or dumps this information in the form of files. Finally, using the saved information, the service can be restored to the original platform or even to a different platform. Through live migration technology, it can also effectively solve problems such as load balancing, system hardware, and software maintenance and upgrade, fault recovery, and rapid service deployment.

The container live migration technology implemented by CRIU [17] can quickly copy and migrate containers by saving the container running state, and realize dynamic load balancing by migrating applications to hosts with less load and meet the needs of high-performance service migration by migrating applications from failed hosts.

However, the current research is not aimed at docker and unnecessary file operations in docker's internal implementation increase the container startup time. Also, the current multi-container migration is only a repetition of the single-container migration, which cannot meet the requirements of dependencies between multiple containers and the need for full resource utilization.

More specifically, the major contribution of this paper includes:

- We propose a method of container migration and a tool for supporting container migration. The tool runs the client of the migration tool on the node where the container or workflow to be migrated and the server on the migration destination host.
- We implement the optimization for starting docker container from checkpoint. By modifying the original code of docker, this paper optimizes the container startup process, reduces unnecessary startup overhead, and greatly improves the performance of the container startup from the checkpoint.
- Aiming at the problem of workflow dependencies and resource utilization, we propose a strategy to migrate containers through pipelines. This strategy controls the migration process in stages, greatly improving the performance of multi-container migration and reduces container migration overhead.

2 Background and Motivation

2.1 Live Migration

In general, there are two ways to migrate virtual machines, memory-based and disk-based. Unlike container live migration, the content saved by virtual machine live migration is relatively simple, and only the physical resources occupied by the virtual machine and the saved state need to be considered. In the field of virtual machines, most of the current popular virtualization solutions provide solutions for live migration [12]. Migration at the entire virtual machine level can use an efficient and easy way to transfer memory, including the state and application level of the internal storage of the kernel State.

There are currently some thermal migration solutions for containers. OpenVZ is an open-source virtualization platform that uses the Virtuozzo kernel. Due to its highly customized kernel, Virtuozzo is one of the first batches of server vendors to provide container live migration solutions. On OpenVZ, the application process checkpoint tool in the kernel is used to save and restore container processes.

A better solution is to accomplish the same thing in the mainstream Linux kernel environment, so Virtuozzo established the CRIU (Checkpoint-restore in Userspace) project team to carry out this effort. As a new solution, CRIU has two advantages over other solutions: 1. This is the only solution that does not require any special operating environment. CRIU runs on a conventional Linux kernel and does not need to recompile the application. 2. This program was originally designed to be used in conjunction with containers, and the traditional checkpoint reduction program did not take this into account [20]. However, there is currently no live migration solution for docker containers. Also, the existing methods have not considered the optimization of multi-container migration.

2.2 Motivation

At present, as an implementation of the more popular container technology, docker integrates the function of starting the container from the checkpoint. This function uses the open-source software CRIU, which is a tool running on the Linux operating system, and its function is to save and restore checkpoints in userspace. Therefore, this function can be used to freeze and restore a single container on the machine.

However, the built-in function of docker cannot meet the needs of migration between different nodes, nor can it meet the live migration needs of multiple containers. Therefore, there is a need for a feasible solution that can implement the multi-container live migration between different nodes. This solution is best based on the process saving and recovery functions provided by CRIU, combined with the docker container management method. In view of the characteristics of high-performance virtualization services, this method can complete the overall live migration of multiple interrelated containers. After the migration is completed, the docker on the destination host should continue to manage and

control the container process like the source host. Finally, we need to optimize performance for application scenarios to minimize migration overhead.

This paper will focus on the feasibility and implementation of container live migration and study the efficient migration solution of multiple containers for different migration scenarios.

3 Overview of Architecture

The design principle of migration tool is driven by two critical factors: building and optimizing the migration tool. These two factors are shown in Fig. 1. The migration tool runs in C/S (client-server) mode, runs the client of the migration tool on the node where the container or workflow to be migrated, and runs the server on the migration destination host. On the destination host, the server of the migration tool is responsible for receiving and processing control messages, sending the information of environment and node status, and completing the creation and recovery of new containers from the received files. On the local machine (source host), the migration tool is responsible for checking the environment, completing container analysis and process dumping, sending files and parameter information needed for container recovery to the server, and controlling the various stages of container migration.

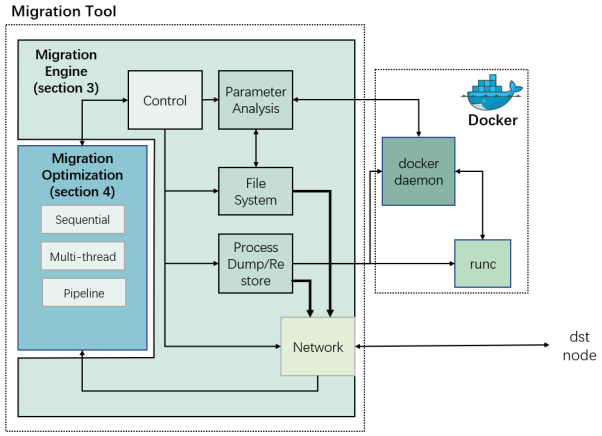

Fig. 1. Architecture of Migration Tool

4 Building Migration Engine

First, we designed a docker container migration engine to handle parameters, processes, and file systems. Second, we modified the source code of docker to avoid unnecessary file operations that reduce the startup time of the docker container.

4.1 Migration engine

As shown in Fig. 1, the main module of migration tool contains four parts.

Parameter analysis. The migration engine needs the parameters of the containers. The engine uses the python library provided by docker to obtain the parameters. As shown in Fig. 1, the migration engine will send the relevant information to the destination host. The server on the destination host receives and creates new containers based on the information to ensure that the configuration of the container has not changed before and after the migration.

Process dump. The migration engine uses runc to dump the process. Runc is a CLI tool that generates and runs containers according to OCI (Open Container Initiative) standards. Runc can obtain all running containers managed by docker. Therefore, the migration engine directly manages the currently running container through runc. Runc calls CRIU through RPC. CRIU is a tool for process dumping and recovery on Linux. It can freeze a running application and persist its state to disk.

File system. In docker, each container is composed of multiple readable mirror layers, and a layer of readable and writable containers will be added when running the container. For the container to be migrated, the migration engine handles the container layered file system in two ways: 1. The engine completes the migration of the readable and writable container layer 2. The existing container layer is packaged into a new image layer, the packaged image is transferred to the destination node, and the container is created according to the image.

Process restore. When restoring the container process, the migration engine calls docker to restore the container process from the checkpoint. Docker calls runc to complete the creation and start of the container. After setting the working environment for CRIU, runc will complete the recovery of the container process by calling CRIU through RPC.

4.2 Modification to docker

At present, the implementation of docker will cause unnecessary overhead during the container startup phase. Docker applies its file reading and writing framework to the dump file. This framework is effective when dealing with small files and facilitates unified management. However, the reading and writing speed under this framework is too slow for large files with hundreds of megabytes. Docker applies a function to output the file system difference between two different directories. This function will output the content in a stream. Docker calls this function to traverse the dump directory and package the contents, and the packaged files will be output to the pipe. After that, docker outputs these contents from the pipe to temporary files in a fixed directory. Contained will read from the previously saved image file through the docker stream, and the read content will be stored by containerd in temporary directory. After completing the operation for the dump file, containerd will prepare for the next call to runc to

complete the recovery of the container process, and the temporary directory will eventually be passed to runc as the new dump file path.

During the test, it was found that docker's way of reading and writing large files takes a long time. The average time of files of about 1GB is between 20 and 30 seconds, which seriously affects the container startup and increases the migration overhead. Therefore, the process of starting the docker container was modified in the migration plan to bypass the process of repeatedly reading and writing files. After modifying the source code of dockerd and containerd and recompiling them, the time overhead caused by repeated file operations is saved. This part is reflected in the experiment in section 4.3 and 6.2.

4.3 Case study: Analysis of container migration time reduction

This experiment tests the migration tool to migrate a single container. Two containers in WGS (Whole Genome Sequencing), wgs indel and wgs call, were used for testing. See section 6.1 for details of the containers.

Case study1: The wgs indel container starts the migration operation of the migration tool when it reaches 600s. At this time, the memory occupied by the container is 1351143424 bytes. As shown in Fig. 2, the total migration time is 44.778s(original) and 22.945s(modified).

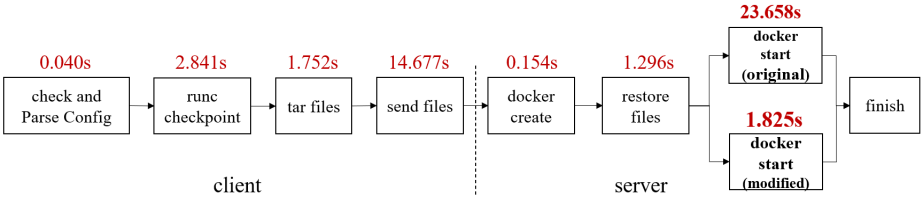

Fig. 2. Wgs indel container migration process

In the same way, the wgs call container was tested, and the test results are as follows:

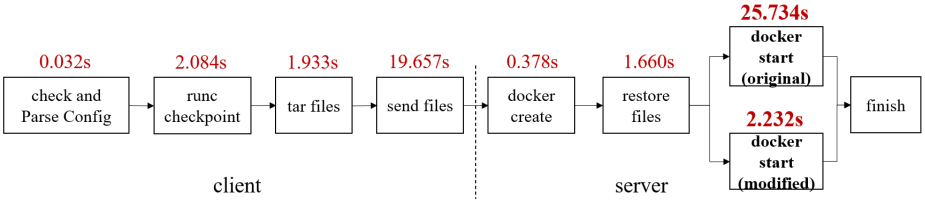

Fig. 3. Wgs call container migration process

As shown in Fig. 4, the modification of docker reduces the migration time by nearly 50% compared to the original version.

Case study2: In addition, we used httpd:2.4.43, mongo:4.2, mysql:8.0.19, nginx:1.17.9, redis:6.0-rc3 for testing and the containers can be successfully recovered. All test containers use the official image in docker hub. Since the time required for migration is closely related to the machine platform, the specific information of the migration process is not listed here.

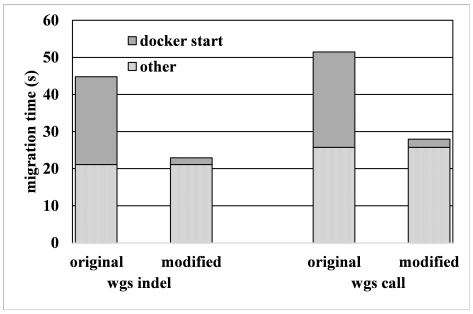

Fig. 4. Comparison of migration time between original and modified version

5 Optimizing Migration Tool

This section proposes a migration solution for scenarios where multiple containers in the workflow need to be migrated. The migration process is controlled in stages and the total migration time is reduced.

5.1 Workflow Migration

Requirements for container order. In a containerized microservices architecture, there are dependencies between services. After being packaged into containers, this dependency is transferred between the containers. To ensure the correctness of the service, it is necessary to carry out the container dump (stop) and the container recovery in a fixed order. Therefore, the migration tool needs to resolve the dependencies between multiple containers to ensure that the containers are stopped and restored in the correct order during the migration process.

Migration overhead of multiple containers. During the container migration process, the migration time overhead generally consists of four parts: process dump, file packaging, network transmission, and process recovery. At each stage, the emphasis on resource utilization is different. When migrating multiple containers, the resource utilization efficiency of the sequential migration method is low. For example, network transmission is only one of four phases. During the execution of the other three phases, the network bandwidth is idle and cannot be fully utilized. Therefore, the migration tool needs to provide a corresponding optimization method in the case of multi-containers to reduce the total migration time of multi-container migration.

In general, users migrate containers in sequence when multiple containers need to be migrated. This approach is not suitable for multi-container migration with complex dependencies. And if the migration process is processed in stages, multiple threads are used to complete the migration, instead of completing them sequentially. The most obvious problem is that it may cause a single container downtime to be too long. The threads started by the migration tool will preempt resources from each other, resulting in a longer container downtime.

In response to the above problems, the migration tool proposes a migration solution for containers in the workflow.

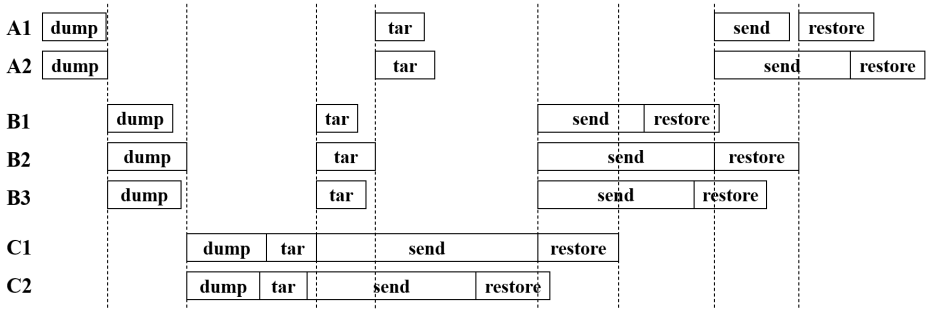

Fig. 5. Schematic diagram of pipeline migration

Pipeline migration. Different from multi-threaded simple parallel jobs, pipeline migration migrates various stages in parallel through the pipeline. The containers of the first start level can be transmitted preferentially on the premise of ensuring the stopping sequence, which reduces the downtime of the containers. To optimize the migration performance, the migration tool adopts a mode that ranks the order of stopping and restoring. The tool does not distinguish the order of containers that are stopped or restored at the same level. This approach ensures that containers with a high startup level will be completed first while avoiding resource competition caused by multiple threads.

To test if our migration method is effective in multi-container migration, we show a comparative case study in Fig. 7 between three methods.

6 Experiment and Evaluation

6.1 Methodology

We mainly used the program in WGS (Whole Genome Sequencing) for testing. WGS sequenced the entire genome of an organic organism to obtain complete genome sequencing information. Multiple containers can be used for WGS together, and each container is responsible for performing one of the steps of gene sequencing. For some of these stages, you can also use multiple containers to complete in parallel. Wgs indel and wgs call were used in the experiment. Wgs indel is responsible for re-alignment in the process, finds the correct area, and performs re-alignment and selection of genetic information. Wgs call is a variant detection link in the WGS process, which mainly deals with genes Mutation site. These two steps are also the most time-consuming in the WGS workflow. In a WGS workflow, multiple indels and calls can be run simultaneously.

6.2 Optimization of Container Startup

This experiment tested the optimization made by the modification to docker during the container startup phase. In this section, the test container used is wgs indel. When the memory occupied by the container is small, for example, when the memory of the container process on the test platform is less than

200MB, there is no significant difference between the original docker and the modified docker. When the memory usage increases gradually, the optimized container startup speed is significantly improved. When the memory usage of the container process reaches more than 700MB, the optimized docker container startup time has dropped to less than one-tenth of the container startup time before optimization.

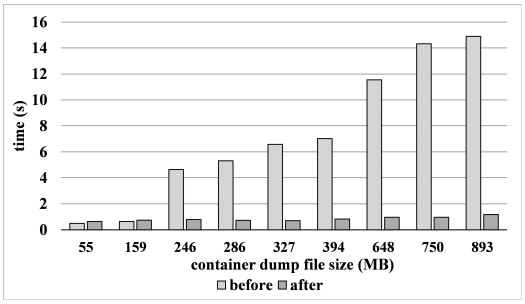

Fig. 6. Optimization of container start from checkpoint

For containers that occupy more than 200MB of memory, using optimized dockerd and containerd programs for container startup will significantly reduce container startup time. For the migration tool, it will also significantly reduce the time taken for container migration and reduce container downtime.

6.3 Test on Workflow Migration

In this section, the three migration methods for multi-container migration in section 3 will be tested. For convenience, these test containers are called A1, A2, B1, B2, C1, and C2, respectively. Among them, C1 and C2 are wgs call containers, and the rest are wgs indel containers. After checking the migration environment, the migration tool officially started. 300s after the startup of these containers, the migration tool is started. Specify the migration order as follows: dump order: first level: A1, A2; second level: B1, B2; third level: C1, C2 startup order: first level: C1, C2; second level: B1, B2 ; third level: A1, A2. The migration order is not considered in sequential migration. The migration process is shown in Fig 7.

Figs. 7(a), (b), and (c)identify the time taken by each container in each stage of the migration process and the time that it is blocked by the migration tool. Among the three migration methods, the total time taken for sequential migration is the longest, reaching 153.8s. The total time for multi-thread migration is 126.4s, which is 18% less than the sequential migration. The total pipeline migration time is 107.7, which is 30% less than the sequential migration. The pipeline method can alleviate the problems of resource competition and insufficient resource utilization, thereby reducing the total time of multi-container migration.

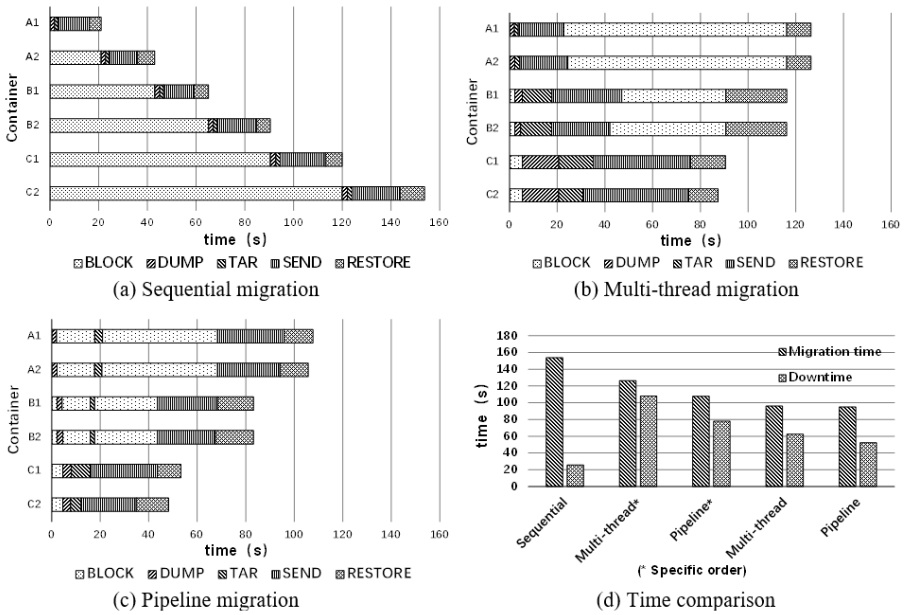

Fig. 7. Time comparison of different migration methods

7 Related Work

Container virtualization technology originally originated on UNIX. By changing the root directory of the process through chroot, it can have the isolation of disk space from the process level. In 2005, OpenVZ was released. This container solution achieved resource management and environmental isolation with the help of a customized Linux kernel. In 2007, Google proposed a common method: Cgroups, used to solve resource management problems on Linux machines [5]. During this period, containers were mainly managed in two aspects, such as resource containers [2, 16] and secure containers [11, 13, 15].

Initially, the namespace on Linux was obtained by the kernel from a single global table for each resource to obtain information. Later, the namespace developed into Plan9-esque [18], each resource of each process can be in a specific namespace [4]. Namespace technology does not provide the function of resource management. Linux provides resource management through Cgroups technology [14].

Process migration was a hot topic in the field of systems research in the 1980s [19, 10, 7, 3], but it was not widely used at the time. The research work on process dump and recovery is currently divided into two categories: system level and application level. Process dumping at the system level was once seen as a more likely solution. This dumping method can be implemented as a general mechanism, and checkpoint and restore operations can be integrated into the cluster resource scheduler [20]. Berkeley Lab's Checkpoint/Restart (BLCR) is a system-level solution consisting of a library and a kernel module, and the appli-

cation must actively provide support for checkpoints [8, 9]. Compared to projects at the system level, process dumping and recovery at the user level generally run on the Linux kernel without special modification, and the operating system does not need special support during the dumping and recovery process. DMTCP (Distributed Multi-Threaded CheckPointing) is a typical user-level application checkpoint restoration project. It implements checkpoint saving and restoration of processes at the library level [1].

Virtuozzo has always been committed to providing real-time migration solutions for virtualization technology. As the first organization to propose live migration of containers, Virtuozzo supports live migration on its containers. However, the implementation method is to integrate the main processes and functions required to perform container live migration into the Virtuozzo kernel, which is a highly customized Linux kernel by Virtuozzo.

A better solution is to accomplish the same thing in the mainstream Linux kernel environment, so Virtuozzo established the CRIU (Checkpoint-restore in Userspace) project team to carry out this effort [20].

The current research is not conducted on docker, and the multi-container migration is not optimized. Based on the current research, this paper proposes a method for docker container migration. Also, this method improves the startup speed of the container and greatly reduces the total time of multi-container migration.

8 Conclusion

This study aims to achieve the live migration of docker containers between nodes, especially the migration of workflows composed of multiple containers. The migration tool implemented in this paper can migrate containers between different nodes, providing a variety of migration options. Also, the tool optimizes the performance of docker containers starting from checkpoints and proposes a optimization solution for multi-container migration in workflow scenarios.

Acknowledgement

This work is supported in part by National Program on Key Research Project (No.2018YFB0204400), by NSFC (No.61702484, No.61972380), by CASSPRP (XDB24050200). Sponsored by CCF-Baidu Open Fund.

References

1. Ansel J, Arya K, Cooperman G. DMTCP: Transparent checkpointing for cluster computations and the desktop[C]//2009 IEEE International Symposium on Parallel & Distributed Processing. IEEE, 2009: 1-12.
2. Banga G, Druschel P, Mogul J C. Resource containers: A new facility for resource management in server systems[C]//OSDI. 1999, 99: 45-58.

3. Barak A, La'adan O. The MOSIX multicomputer operating system for high performance cluster computing[J]. Future Generation Computer Systems, 1998, 13(4-5): 361-372.
4. Bhattiprolu S, Biederman E W, Hallyn S, et al. Virtual servers and checkpoint/restart in mainstream Linux[J]. ACM SIGOPS Operating Systems Review, 2008, 42(5): 104-113.
5. Clark J. Google:'EVERYTHING at Google runs in a container'[OL]. https://www.theregister.co.uk/2014/05/23/google_containerization_two_billion
6. Docker[OL]. https://docs.docker.com/get-started/overview/
7. Douglis F, Ousterhout J. Transparent process migration: Design alternatives and the Sprite implementation[J]. Software: Practice and Experience, 1991, 21(8): 757-785.
8. Duell, Jason. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart[J]. lawrence berkeley national laboratory, 2005.
9. Hargrove P H , Duell J C . Berkeley lab checkpoint/restart (BLCR) for Linux clusters[J]. Journal of Physics Conference, 2006, 46:494-499.
10. Jul E, Levy H, Hutchinson N, et al. Fine-grained mobility in the Emerald system[J]. ACM Transactions on Computer Systems (TOCS), 1988, 6(1): 109-133.
11. Kamp P H, Watson R N M. Jails: Confining the omnipotent root[C]//Proceedings of the 2nd International SANE Conference. 2000, 43: 116.
12. Kotikalapudi S V N. Comparing live migration between linux containers and kernel virtual machine: investigation study in terms of parameters[J]. 2017.
13. MacCarty B. SELinux-NSA's open source security enhanced linux: beating the o-day vulnerability threat[J]. 2005.
14. Menage P B. Adding generic process containers to the linux kernel[C]//Proceedings of the Linux symposium. 2007, 2: 45-57.
15. Morris J, Smalley S, Kroah-Hartman G. Linux security modules: General security support for the linux kernel[C]//USENIX Security Symposium. ACM Berkeley, CA, 2002: 17-31.
16. Nagar S, Franke H, Choi J, et al. Class-based prioritized resource control in Linux[C]//2003 Linux Symposium. 2003.
17. Pickartz S, Eiling N, Lankes S, et al. Migrating LinuX Containers Using CRIU[C]. ieee international conference on high performance computing, data, and analytics, 2016: 674-684.
18. Pike R, Presotto D, Thompson K, et al. The use of name spaces in Plan 9[J]. Operating Systems Review, 1993, 27(2): 72-76.
19. Powell M L, Miller B P. Process migration in DEMOS/MP[J]. ACM SIGOPS Operating Systems Review, 1983, 17(5): 110-119.
20. Vasyukov A, Beklemysheva K. Using CRIU with HPC Containers Field Experience[J]. International Journal of Engineering and Computer Science, 2018, 7(07): 24106-24108.