Sledge : Towards Efficient Live Migration of Docker Containers

Bo Xu*, Song Wu*, Jiang Xiao*, Hai Jin*, Yingxi Zhang†, Guoqiang Shi†, Tingyu Lin†, Jia Rao‡, Li Yi§, Jizhong Jiang§
*National Engineering Research Center for Big Data Technology and System
Services Computing Technology and System Lab, Cluster and Grid Computing Lab
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China
† State Key Laboratory of Intelligent Manufacturing System Technology, Beijing, 100854, China
‡The University of Texas at Arlington, Arlington, TX 76019, USA
§Alibaba Cloud Computing Co. Ltd., Hangzhou, 310012, China
Email: wusong@hust.edu.cn

***Abstract*—Modern large-scale cloud platforms require live migration technique on Docker containers with stateful workload to support load balancing, host maintenance, and *Quality of Service* (QoS) improvement. Efficient and scalable Docker live migration is expected to guarantee the component-integrity (image, runtime, and management context) with negligible downtime. In this paper, we present a highly efficient live migration system called *Sledge*, which ensures the component-integrity by integrating both images and management context during runtime migration. The key insight is that the layered image can be leveraged to reduce the migration overhead, and appropriately selective migration of management context will effectively improve QoS with negligible downtime. To achieve good scalability, a lightweight container registry mechanism for end-to-end image migration is designed to avoid the redundant layers transmission. In addition, a dynamic context loading scheme is proposed to precisely load the management context into the running daemon, which can significantly reduce downtime. Experiments show that, compared with the state-of-the-art, *Sledge* reduces 57% of total migration time, 55% of image migration time, and 70% downtime.**

***Keywords*-Docker, Live Migration, Container, Cloud**

I. Introduction

Docker [1] makes it convenient for developers to package the application runtime into an image, and run the application on any OS with the assistance of Docker daemon. In a large-scale data center, millions of Docker containers are usually managed by various orchestration tools (i.e., Kubernetes [2], Mesos [3], and Swarm [3]). None of them can fulfill the live migration requirements for Docker containers in the scenes of load balancing, host maintenance, and system upgrade. Kubernetes leverages replicate controller to relocate Docker containers as an alternative solution. It can seamlessly move stateless Docker container to another node, but will bring terrible downtime for stateful Docker containers which are more and more popular in cloud environments. As a result, live migration of Docker container is a desirable and valuable technology for resource utilization and QoS guarantee in data centers.

There are mature works [4]–[8] and migration mechanisms (*i.e.*, Pre-Copy [9], Post-Copy [6], and Logging-Replay [10]) for live migrations of *virtual machines* (VMs). Different from VM, Docker has a layered image, a shared-kernel runtime, and a richer functional management architecture, which make live migration of a Docker container more complicated. Correspondingly, live migration of Docker containers consists of three-part tasks, i.e., migration of image, migration of runtime, and migration of management context. During the procedure of live migration, it is undoubtedly important to guarantee the integrity of three key components, also known as component-integrity. Besides, the scalability and downtime of the live migration are the critical metrics in a data center. As a result, ideal live migration of Docker containers should provide good scalability and negligible downtime in performance, while guaranteeing the component-integrity of Docker containers in terms of functionality.

However, it is difficult to meet these goals. In functionality, most related works overlook the component-integrity and only focus on the migration of runtime [11], which is as accordance with the *Open Container Initiative* (OCI) as other containers (i.e., LXC [12] and CoreOS Rocket [13]). For example, P. Haul [14] and Voyager [15] migrate image by only synchronizing the *rootfs* between nodes. On the one hand, they cannot leverage the layered feature of images to avoid redundant layers transmission. On the other hand, they cannot benefit from the promising features of the layered image that enables fast packaging and shipping of applications, such as version-control and layer sharing. Further, they lack the consideration of management context and lose the management of life-cycle, images, network, and volumes for migrated Docker container, which are harmful to data center management [3].

Ma et al. [16] guarantee the component-integrity of Docker containers for the scene of edge servers during live migration, but becomes inappropriate in a data center due to poor scalability and intolerant downtime. They pull images from a central image registry for image migration, which consumes huge bandwidth of image registry and violates migration performance. In addition, the procedure of reloading daemon to restore management context leads to a second-level waste of downtime owing to needless

2159-6190/20/$31.00 ©2020 IEEE
DOI 10.1109/CLOUD49709.2020.00052

initialization.

In this paper, we present Sledge, a highly efficient live migration system for Docker containers. Sledge guarantees the components-integrity, high scalability, and negligible downtime during live migration in data centers. In particular, we first design the *lightweight container registry* (LCR) mechanism to achieve an end-to-end image migration. Based on in-depth analysis of the hierarchy feature of image layers, LCR can effectively avoid the redundant transmission of existing layers. Second, we iteratively migrate Docker container's runtime by incorporating CRIU's [17] incremental memory checkpointing capabilities. Third, we propose *dynamic context loading* (DCL) scheme by exploiting the coupling relationship between Docker daemon and Docker containers. DCL can precisely load the management context of a Docker container into running daemon, which significantly reduces downtime by avoiding reloading Docker daemon.

This paper makes the following contributions:

1. We design and implement an efficient live migration system, Sledge, to support component-integral and scalable live migration for Docker container in cloud platforms. To the best of our knowledge, Sledge is the first system that can meet both functional and performance requirements.

2. We present a novel LCR mechanism for image migration, which can avoid redundant layers transmission and improve scalability. We further design DCL scheme for management context migration. DCL makes it more flexible to restore the management context of Docker containers, which can reduce the migration downtime and have no interference to the running Docker containers on target node.

3. Our evaluation with various images shows that Sledge outperforms state-of-the-art [16] by 57% lower total migration time, 55% lower image migration time, and downtime by 70%, on average.

II. Motivation

It is common in a data center that we need to relocate Docker containers for the reason of load balancing, host maintenance, and system upgrade. Stateless Docker containers are easy to be smoothly relocated between physical nodes. As Kubernetes has done, it shutdowns a stateless Docker container and launches the preliminary replicated one to complete the relocation. However, it is not applicable for stateful Docker containers, such as those with database and online game applications. The states of a Docker container has to be saved in volumes and then synchronized to the replicated Docker container, which probably generates terrible downtime. Consequently, an efficient live migration of Docker container is desirable in data centers.

Since the migration of Docker containers is more complicated than VMs or processes [18], hereby we analyze the migration from three key aspects.

A. Image Migration

There are two traditional ways to implement image migration. One is to pull image from *centralized image registry* (CIR) and then synchronize mutable read-write layers. The other is migrating image by using native interfaces of Docker (ie., "docker save" and "docker load").

However, both ways are not suitable for live migration. The former makes the bandwidth of CIR become the latent bottleneck of image migration performance. The latter one suffers from numerous [19], massive image migrations with frequent image pullings from CIR will surely be harmful to the global migration performance in data centers.

Alternatively, we propose the LCR mechanism based on the hierarchical features of image layers to implement end-to-end image migration while avoiding transmission of redundant layers.

B. Runtime Migration

In order to reduce downtime during migration, it is necessary to iteratively synchronize memory footprints. CRIU [17] is a popular and indispensable technology for live migration of processes inside Docker containers. Therefore, we incorporate CRIU incremental memory checkpointing ability and apposite iterative synchronization algorithm for Docker containers to achieve the goal.

C. Context Restore

Docker daemon loads related information from local storage to build management context only during initialization. In order to keep the consistency of runtime and restore the management context of migrated container, the traditional way is to reload (i.e., kill and restart) Docker daemon on target node. However, reloading daemon seriously increases migration downtime and affects the performance of running containers on target node.

To eliminate the latency caused by reloading daemon, we design DCL scheme to load the management context of migrated containers into the running daemon of target node.

III. Preliminary

A. Docker Image

A Docker container has a read-only base image and a *read-write layer* (RWLayer), which will be combined into a directory as the *rootfs* to provide a global perspective of container file system. When a container attempts to write a file, the file will be copied to the top read-write layer so that read-only layers can be shared by different images. In general, layered image can facilitate fast packaging and shipping of applications [20], which makes it convenient to perform *continuous integration* and *continuous deployment* (CI/CD) of applications.

B. Docker Daemon

Docker daemon plays the role of server that maintains and manages all container related information, including metadata, image, volume, network, as the core of Docker architecture. The initialization of Docker daemon not only loads the information to build management context, but also involves the start-up of the other components. As a result, it will result in second-level delay to reload Docker daemon.

The daemon manages Docker containers mainly based on the following four drivers: Layers Driver recording the layer related information such as storage path and reference time, Container Driver recording configurations and characteristics of containers, Image Driver recording image environment configuration information, and Volume Driver recording the information of container related data volumes.

Docker daemon effectively manages containers and images by obtaining any information it wants from the four drivers. The four drivers provide various interfaces to manage containers based on the stored data. As we can see, loading Docker container management context is only a little part of the whole procedure of reloading daemon. Therefore, it is not worthy to reload Docker daemon for the purpose of restoring management context of migrated Docker container.

IV. Design and Implementation

Hereafter, we will first summarize the design goals of Sledge and then we focus on details of the design and implementation of Sledge.

A. Design Goals

1) Component integrity: Complete live migration of Docker container involves three parts: runtime, image, and management context. Unlike most existing work only addressing container runtime migration, Sledge must take all of the three parts into consideration, which makes the solution integral and practical. Therefore, the layered image feature and powerful management context of a Docker container can be completely maintained after live migration.

2) High scalability: An efficient solution of Docker container live migration should be highly scalable to avoid possible performance bottleneck especially in a large-scale data center. It should reduce dependence on the centralized container image registry and decrease data transfer during live migration, which can guarantee whole system performance in face of massive container live migrations. Sledge will design a distributed and lightweight container registry architecture to achieve an end-to-end image migration, and leverage the layered feature of Docker images to reduce the unnecessary data transmission during image migration.

3) Minimized downtime: Downtime is the key performance metric of live migration system, which has an important impact on QoS of docker-hosted applications. Currently, Docker live migration downtime mostly comes from context reloading instead of incremental memory copy during live migration. Sledge aims to remarkably lower the migration downtime of Docker containers by introducing a dynamical seamless context restoring method.

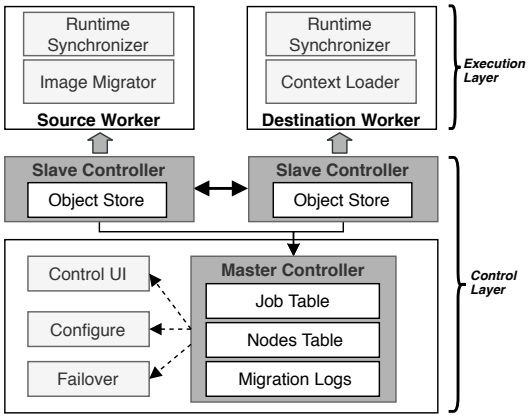

Figure 1. *Sledge* consists of two layers: control layer and execution layer

B. Design of Sledge

1) Overview: Sledge consists of two layers, control layer managing the migration events in a cluster, and execution layer implementing the procedure of migration, as shown in Fig. 1.

In control layer, Master Controller distributes migration jobs to Slave Controller and records migration logs. Slave Controller runs on each physical node, keeps listening to the migration request from Master Controller, and continuously synchronizes migration status. It derives source worker or destination worker based on migration request to play different roles in the migration process.

In execution layer, Source Worker migrates image and runtime by Image Migrator and Runtime Synchronizer. Besides, it records migration logs and maintains the relevant information of migrated Docker container. Destination Worker participates in runtime synchronization by Runtime Synchronizer and restores management context by Context Loader.

2) LCR-based Image Migrator: Image Migrator adopts *lightweight container registry* (LCR) mechanism to accomplish image migration. The primary design for Image Migrator is to fulfill three properties: locality, plug-and-play, and lightweight.

LCR adequately leverages local image data. As shown in Fig. 2, relying on the comprehensive analysis of local image layer hierarchy, LCR can integrate and maintain the image, meta and volume data of migrated Docker container.

For the plug-and-play property, it is not expected to run another backstage process to maintain all of the local image information. We should avoid generating much extra overhead on the source. Therefore, Image Migrator establishes

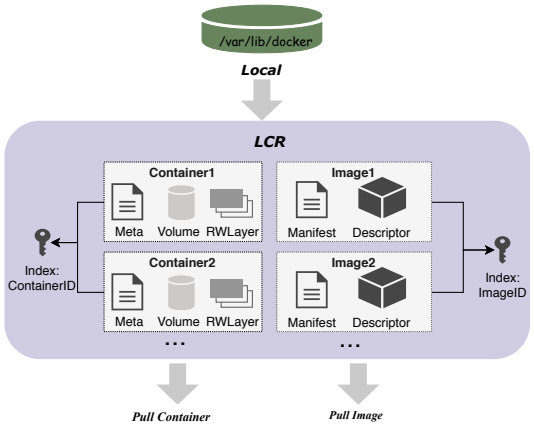

Figure 2. *LCR* consists of two data sets: image and container. An image is divided into layers which are recorded by *manifest* and addressed by *descriptor* at local. The relevant information of a container except its image can also be addressed by *containerID* at local.

lightweight container registry only when image migration is triggered and immediately destroyed after migration.

Establishing and destroying lightweight container registry is part of the total migration time, so that container registry should be as lightweight as possible to further confine the extra overhead. Compared to traditional image registry, our lightweight container registry cuts out most unnecessary functions (e.g. search, push, and delete), and only retains pull operations for image migration.

In summary, LCR-based Image Migrator puts image registry at local to improve the scalability of migration, and simplifies image registry to strive for lightweight and avoid extra overhead.

3) Pre-Copy Runtime Synchronizer: Downtime is the most important metric to evaluate the performance of live migration. Sledge adopts the most acknowledged migration algorithm, Pre-Copy, to accomplish the runtime migration of Docker container. To implement Runtime Synchronizer, we analyze the runtime of Docker containers and integrate the incremental checkpoint ability of CRIU into Docker. The runtime of Docker container is composed of the real-time status (mainly memory) of the processes in the container, the read-write layer of Docker container, and the volume. The migration procedure is designed as Fig 3.

Runtime Synchronizer of Sledge focuses on the migration of Docker container instance and defaults the volume as the remote solution. We only need to disconnect the volume from the source and reconnect the volume to the destination before and after the *stop-and-copy*. The read-write layer of a Docker container usually has little change in production environment and can be incrementally synchronized by the tool of ”rsync”. Therefore, we design stop-and-copy conditions of Pre-Copy algorithm for Docker container as shown in Fig 4. Initially, we will get a snapshot of the migrated container memory footprints by CRIU, which is saved into checkpoint files. After that, the checkpoint files will be transferred to the target’s disk. Finally, memory in checkpoint files is read into the target’s memory to restore the real-time state of migrated container. Let T represent the max tolerable downtime, N represent the network bandwidth of the data center, and D represent the disk IO rate. We calculate the threshold size of the memory checkpoint file:

$$S = T * (\frac{DN}{2N + D}) \tag{1}$$

When the size of incremental memory checkpoint file in consecutive iterations goes under the threshold S MB, we should start *stop-and-copy*. In Sledge, the basis for judging whether the iterative process is divergent is whether the size of incremental memory in this iteration is larger than the last iteration. The maximal number of iteration times defaults to be 10 in Sledge.

4) DCL-based Context Loader: Reloading Docker daemon results in a terrible downtime for live migration procedure, which is mainly blamed on unnecessary initialization. Docker daemon manages local containers in an active way and only loads the information of local Docker containers from storage during the procedure of initialization. To minimize downtime, we design DCL-based Context Loader to break the barrier of Docker daemon and local storage, which can achieve the bottom-up management of Docker containers. It enables Docker daemon to passively perceive the advent of migrated container from local storage environment. Furthermore, the management context of migrated container can be precisely loaded into running daemon to avoid time-consuming initialization, relying on the analysis of coupling relationship between Docker daemon and containers.

To guarantee the reliability and security of dynamic context loading, DCL scheme should have the following three characteristics. First, DCL is designed to have self-validate ability to ensure the integrity and validity of migrated Docker container. Second, to make entire loading process controllable, DCL should have self-memory ability to support back-track operation. Third, DCL should enable self-clean to prevent potential crashes of Docker daemon,

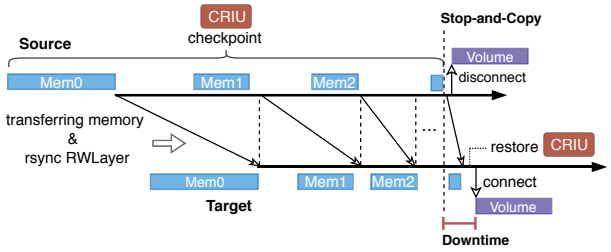

Figure 3. Migration procedure. To reduce downtime, we leverage *CRIU* to iteratively checkpoint incremental memory of container until the memory size reaches the threshold.

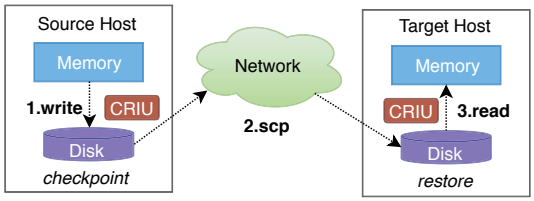

Figure 4. Memory migration. Each memory migration involves three steps: writing to a disk, transferring to target, and reading from a disk.

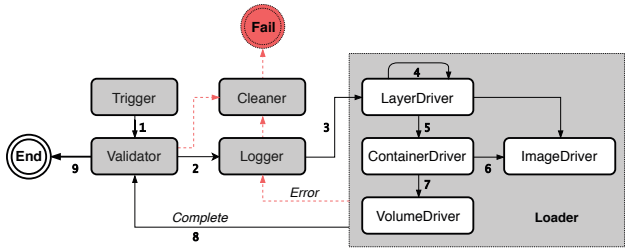

Figure 5. Architecture and workflow of DCL. The coordination of these components can guarantee that the procedure of loading migrated container information has no interference to the target node.

due to data inconsistency brought by unpredictable errors during loading procedure.

To this end, we implement five key components including *Trigger*, *Validator*, *Loader*, *Logger*, and *Cleaner*. Trigger is in charge of checking if the trigger condition is reached and starting the subsequent process. Validator can verify the feasibility and integrity of loaded data in drivers. Loader loads the information needed by daemon from local data in correct order. During loading process, Logger records the loading history. The last component, Cleaner, cleans the information that has been loaded into the daemon by the Loader when some errors have occurred in Loader or Validator. Figure 5 depicts the design and workflow of the modules.

C. Putting Everything Together

The overall migration procedure of Sledge can be decomposed into two workflows: control flow and data flow.

Control flow. Slave Controller joins into Master Controller, and keeps synchronizing migration log with the master. Master Controller can create jobs and distribute them to corresponding Slave Controller. Slave Controller will parse the requirements and derive Source or Destination Worker to perform different tasks. Meanwhile, the Source Worker and Destination Worker establish a connection to coordinate during migration procedure.

Data flow. Firstly, Source Worker establishes a lightweight container registry on the source. Destination Worker will pull image from LCR and adjust the hierarchical relationship of image layers. Secondly, Runtime Synchronizer migrates the runtime according to the Pre-Copy algorithm. Finally, the management context will be loaded into the Docker daemon dynamically. The running daemon can restore the ability to manage the migrated Docker container. In this way, stateful Docker container succeeds in migrating to another node.

V. Evaluation

We simulate performing live migrations of Docker containers in a cluster with 20 nodes. Each node is equipped with Ubuntu 16.04 LTS as operating system, 2 CPU cores, 2GB memory, and 50GB disk with ext4 filesystem. Besides, network bandwidth between nodes can reach 500Mbps. To enable CRIU incremental checkpoint, we modify the source code of Docker version 1.13 by adding two options (i.e. , *–predump* and *–parent*). In particular, the version of CRIU is 3.10.

We evaluate Sledge using metrics including migration performance, scalability, and downtime. A single Docker container is migrated five times between two nodes so as to figure out the average. The experiments employ two types of Docker container workloads:

Stateless container: Containers are built on images with most download frequency in Docker Hub, which are Busybox, Mysql, Nginx, Redis, and Ubuntu. In order to make these containers stateless, no load will run on them. Therefore, they consume little memory footprint.

Stateful container: We select Apache container presented by SPECweb99, a benchmark for evaluating the performance of a web server. In addition, we select Vsftpd container as the FTP server.

A. Migration Performance

To compare Sledge against prior Docker migration system, we perform the migrations of above workloads by Sledge, and most related work LMM [16], and Voyager [15], respectively.

Mysql and Ubuntu are selected as representatives of stateless containers. In addition, we arrange 10 clients, each of which sends a random request to the Apache server per second. We also have a client sending files to Vsftpd server with the network throughput as 50Mbps. Prior to the migration, we placed about 50% redundant layers cache of above containers on the target. The bandwidth of the centralized image registry is 2Gbps, and the bandwidth between the source and target is 500Mbps. The total migration time and total network throughput are recorded.

First, we concurrently perform 10 migrations for the above workloads on the cluster with 20 nodes. We compare the total migration time of different migration solutions. From Figure 6, Sledge completes these migrations in the shortest time, while LMM has the worst performance. The reason is that Sledge can achieve the decentralized migration while LMM has to be bothered by the bandwidth bottleneck of image registry. According to our experiments, Sledge

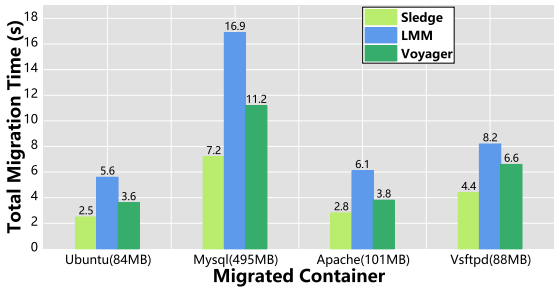

Figure 6. The total migration time comparison among Sledge, LMM, and Voyager when concurrently performing 200 migrations for different containers

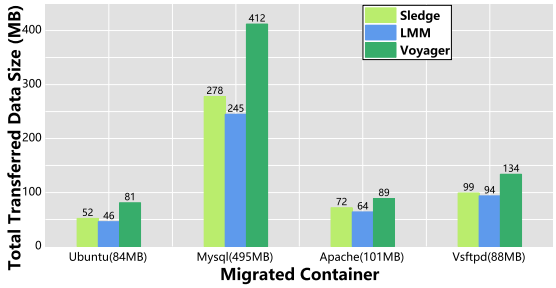

Figure 7. The total transferred data size comparison among Sledge, LMM, and Voyager when performing one migration for different containers

can reduce migration time by 57% when we concurrently migrate 10 Mysql containers.

Secondly, we perform one migration of above workloads on 2 nodes to find out the amount of total transferred data. From Figure 7, the transferred data size of Sledge is slightly higher than LMM but significantly lower than Voyager, because Sledge and LMM can reuse the image cache on the target to avoid the redundant layer transmission.

B. Scalability

As discussed above, LMM makes the bandwidth of registry become a bottleneck for migration. Thus, they scale poorly for the numerous concurrently image migration. We conduct two experiments to verify the scalability of Sledge, whose results are shown in Figure 8 and Figure 9.

In Figure 8, the x axis represents image sizes which are

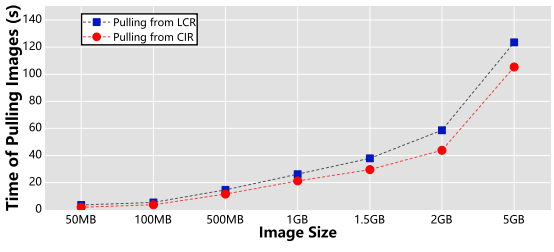

Figure 8. The comparison of pulling image from LCR and centralized image center (CIR). The image size ranges from 50MB to 5GB

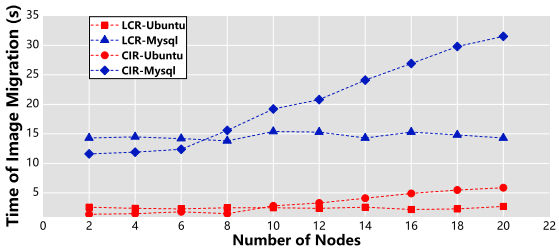

Figure 9. The comparison of migrating Ubuntu image and Mysql image by pulling from LCR and CIR with many nodes whose number ranges from 2 to 20

between 50MB and 5GB, while y axis represents the time of pulling images. There are two lines which respectively react to the time increasing tendency as the image size increases in Sledge and LMM. Sledge has worse performance in pulling images from LCR, because it needs to wait for LCR to package image layers into tar files. Compared to pulling images from *central image registry* (CIR), Sledge has to spend more time on packaging.

We try to scale from 2 nodes to 20 nodes. We set up an image registry equipped with 2Gbps bandwidth. We simultaneously migrate Ubuntu image (84MB) and Mysql image (495MB) to test the time of completing the migrations on all nodes. From Figure 9, we can find that the time of pulling images from CIR increases linearly as the number of nodes increases after the number of nodes reaches 6. By contrast, Sledge experiences little increases of time. According to our experiments, Sledge can reduce image migration time by 55%, when we migrate Mysql image and the number of nodes is increased to 20. Therefore, we can conclude that Sledge has a better performance in a data center than LMM because the LCR of Sledge can obviously improve the scalability of image migration.

C. Downtime

This subsection will test the downtime of the aforementioned stateless and stateful containers. Both network bandwidth between the two nodes and disk IO rate on the physical host are set as 500Mbps. The maximum tolerable downtime is set as 1s. Therefore, the threshold of memory size is about 30MB according to Equation 1.

1) Stateless Container: From Figure 10, we can see downtime can be controlled less than 1s, and Sledge's *dynamic context loading* (DCL) scheme reduces downtime by 74% for Busybox, 74% for Redis, 75% for Nginx, 74% for Mysql, and 77% for Ubuntu compared to LMM. Therefore, according to our experiments, Sledge reduces downtime by 75% on average for stateless containers.

2) Stateful Container: Apache container receives various requests, including dynamic standard GET operation, dynamic random GET operation, dynamic POST operation and so on. Vsftpd container deals with requests and transfers files between client and server. During the process of live

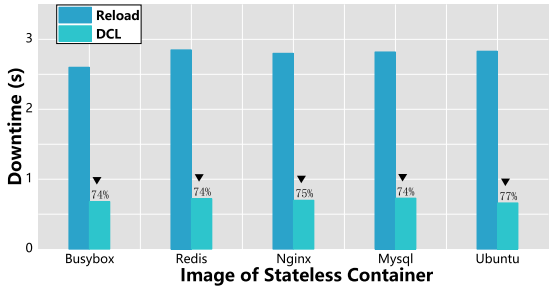

Figure 10. Downtime comparison for five stateless containers

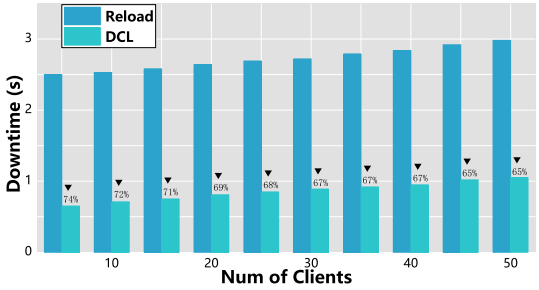

Figure 11. Downtime comparison of web server which is receiving requests from many clients

migration, we put the two nodes into the same overlay network, so that client can discover service by the same IP address.

Web Server: To evaluate the downtime of web server container, clients are set to send requests to Apache server simultaneously, whose number ranges from 5 to 50 with step size as 5. Each of the clients sends a random request once per second. From Figure 11, we can see that Sledge's DCL can reduce downtime obviously, and downtime of the two ways increase less as the number of clients increases. In general, it is calculated that downtime has been reduced by 69% for Apache server on average based on our experimental data.

FTP server: We also perform migrations in two ways for Vsftpd container, whose network throughput ranges from 5MB/s to 50MB/s with the step size as 5MB/s. It can be concluded from Figure 12 that downtime can be reduced obviously by Sledge's DCL. According to our experiments, downtime is reduced by 65-74%. We can observe that there is a decrease when the throughput reaches 35MB/s. The reason is that the memory size exceeds the threshold, thus Sledge tries to reduce downtime by iterative incremental memory migration.

In summary, Sledge can reduce downtime during live migration by 70% on average compared to LMM.

VI. Related Work

The live migration of container is a desirable technology in container clouds. Up to now, some researchers have put massive efforts into the live migration of a container.

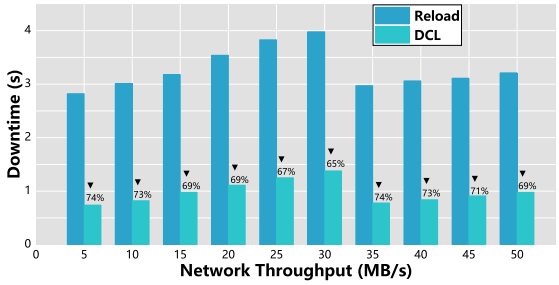

Figure 12. Downtime comparison of FTP server which is transferring files with different network throughput

Virtuozzo [21] can support container virtualization in a bare-metal architecture and also can perform the live migration of container. However, in the previous versions of Virtuozzo, most operations performed during migration were done in kernel space. As a result, the migration of process imposed a lot of restrictions. To improve migration, Virtuozzo launched CRIU [17], aiming to move most of the migration code to user space, making the migration process reliable to support live migration of Virtuozzo. Up to now, CRIU has drawn extensive attention and become the key technology for container live migration. But the live migration of Docker differs largely from Virtuozzo, except that they all need to migrate process with the assistance of CRIU.

Flocker [22] is an open-sourced volume management tool for Docker and supports the migration of volume. Flocker makes it possible to run a stateful container for operation maintenance because the volume of a container can be migrated between any nodes in container clouds. But Sledge is a more comprehensive solution for the live migration of a Docker container, which not only supports migration of volume but also supports the whole execution environment.

Both Voyager [15] and P. Haul [14] implement the runtime live migration. But they are not enough for a Docker container. Voyager implements the live migration of runtime base on Post-Copy and filesystem federation. P.Haul is a project launched by OpenVZ [17] that can implement the external migration of a Docker container. External migration means that P.Haul only implements the live migration of process in a Docker container by CRIU. These works cannot guarantee the component-integrity of Docker containers.

Ma et al. [16] complete the three-part tasks on edge servers, which pulls image from CIR and then restores hierarchical relationships by ID remapping. But this way is not practical in container clouds, because pulling images from CIR is excessively dependent on the bandwidth of centralized image registry, which puts too much pressure on the core network. They migrate the runtime of container by only one iteration of memory checkpointing, which can not guarantee that downtime is short enough. After that, they adopt the original and straightforward way to make

Docker daemon be aware of the migrated container by reloading Docker daemon. However, reloading daemon is time-consuming and has a serious impact on downtime. Therefore, their way cannot meet the performance requirements of scalability and downtime.

As we can see from above, Voyager [15], P. Haul [14], LMM [16], and Sledge can support live migration of a Docker container to some extent. Note that only Sledge can satisfy both the functionality and performance requirements of live migration for Docker containers relying on the properties of layered image and management context in a data center.

VII. Conclusion and Future Works

No prior study about the live migration of Docker containers can simultaneously guarantee component-integrity, good scalability, and negligible downtime. In order to facilitate a thorough and comprehensive migration in a data center, Sledge leverages LCR to migrate a Docker image in an end-to-end manner, which markedly improves scalability. LCR is constructed based on the comprehensive analysis of local image layers' hierarchical relationship, and it can effectively alleviate the bandwidth pressure of central image registry in a data center. Moreover, Sledge has incorporated CRIU incremental memory checkpoint ability to iteratively synchronize memory and control downtime under threshold. Finally, according to DCL scheme, Sledge implements a dynamical loading module for management context migration, which obviously reduces downtime by eliminating the latency caused by reloading daemon. From experimental results, we can see that Sledge has the best performance in a container platform with numerous concurrent migrations.

Our future work will focus on Kubernetes pod migration. It is desirable to organize containers with pod as a snap-in by Kubernetes. In a pod, multiple Docker containers share network, volume, and local image cache. We will implement pod migration based on Sledge.

VIII. Acknowledgment

This work is supported by National Key Research and Development Program under grant 2016YFB1000501, National Science Foundation of China under grant No.61732010, and Pre-research Project of Beifang under grant FFZ-1601.

References

[1] C. Anderson, "Docker," *IEEE Software*, vol. 32, no. 3, pp. 102–c3, 2015.

[2] *Kubernetes.*, https://kubernetes.io/.

[3] R. Peinl, F. Holzschuher, and F. Pfitzer, "Docker cluster management for the cloud-survey results and own solution," *Journal of Grid Computing*, vol. 14, no. 2, pp. 265–282, 2016.

[4] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of NSDI'05*, pp. 273–286.

[5] S. Nathan, U. Bellur, and P. Kulkarni, "On selecting the right optimizations for virtual machine migration," in *Proceedings of VEE'16*, pp. 37–49.

[6] M. R. Hines and K. Gopalan, "Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning," in *Proceedings of VEE'09*, pp. 51–60.

[7] H. Jin, L. Deng, S. Wu, X. Shi, H. Chen, and X. Pan, "MECOM: live migration of virtual machines by adaptively compressing memory pages," *Future Generation Computer Systems*, vol. 38, pp. 23–35, 2014.

[8] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *Proceedings of HPDC'11*, pp. 171–182.

[9] K. Z. Ibrahim, S. Hofmeyr, C. Iancu, and E. Roman, "Optimized pre-copy live migration for memory intensive applications," in *Proceedings of SC'11*, pp. 40–50.

[10] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu, "Live migration of virtual machine based on full system trace and replay," in *Proceedings of HPDC'09*, pp. 101–110.

[11] S. Pickartz, N. Eiling, S. Lankes, L. Razik, and A. Monti, "Migrating Linux containers using CRIU," in *Proceedings of ISC'16*, pp. 674–684.

[12] D. Bernstein, "Containers and cloud: from LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[13] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[14] R. Boucher, *P.Haul*, https://github.com/boucher/p.haul.

[15] S. Nadgowda, S. Suneja, N. Bila, and C. Isci, "Voyager: complete container state migration," in *Proceedings of ICDCS'17*, pp. 2137–2142.

[16] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via Docker container migration," in *Proceedings of SEC'17*, pp. 11–23.

[17] *CRIU*, https://criu.org/Main_Page.

[18] S. Pickartz, J. Breitbart, and S. Lankes, "Implications of process-migration in virtualized environments," in *Proceedings of COSH'16*, pp. 31–36.

[19] C. Kulkarni, A. Kesavan, T. Zhang, R. Ricci, and R. Stutsman, "VM Live Migration At Scale," in *Proceedings of VEE'18*, pp. 45–56.

[20] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: fast distribution with lazy docker containers," in *Proceedings of FAST'16*, pp. 181–195.

[21] *OpenVZ.*, https://openvz.org/Main_Page.

[22] *ClusterHQ. Flocker.*, https://docs.clusterhq.com/en/1.0.3/.