# 國 立 清 華 大 學

資訊工程研究所

碩士論文

# 使用中斷點和回復機制
# 實現容器叢集中的遷移和高可用性
# Container Migration and High Availability in Docker Swarm using Checkpoint and Restoration

研究生：103062622 黃晟豪（Cheng-Hao Huang）

指導教授：李哲榮 教授（Prof. Che-Rung Lee）

中華民國一零五年七月

# Container Migration and High Availability in Docker Swarm using Checkpoint and Restoration

**Student: Cheng-Hao Huang**
**Advisor: Prof. Che-Rung Lee**

**Department of Computer Science**
**National Tsing Hua University**
**Hsinchu, Taiwan, 30013, R.O.C.**

July 2016

# 中文摘要

容器技術自 2013 年 Docker 發表後在全世界迅速竄紅,Container 解決了維護人員在伺服器進行大量部屬時的痛點，使得環境部屬只需要建立完容器映像檔後就可以進行大量部屬，並對每一個容器環境進行隔離。

在這篇論文中，我們提出了在 Docker swarm 叢集中，將容器在多個節點中相互搬移。另外，可以針對特定的容器定期設定 checkpoint 儲存至雲端儲存空間，若叢集中的節點遇到不正常的離線時，可以及時回復最近的容器狀態到健康的節點上。

## Abstract

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# Chapter 1

# Introduction

Traditionally, people rely on a single supercomputer to calculate data or deploy their companies' applications. Nowadays, cloud computing has been used widely. People would like to use a lot of normal computers and gather them into a cluster to replace a supercomputer. An increasing number of companies build their data centres or use cloud platforms to construct their business application. However, the more computers we have, the larger of a power consumption problem we have to solve. At the same time, to make sure every computers' process in the cluster are alive, high availability becomes a more important role in cloud computing.

Virtualization is a popular technology that is used wildly in cloud computing, including virtual operating systems, computer hardware platforms, storage devices and computer network resources. Virtual machine is a technology to emulate a particular computer like a real computer. It can partition the physical machine's resources, such as CPU, memory, storage, and network. A hypervisor uses execution to manage and share the host physical machine hardware, it allows many different virtual machines to be isolated from each other.

Container is an operating system level virtualization which runs as an isolated process in userspace on the host operating system and shares the same operation system kernel with other containers. It provides kernel namespace such as PID, IPC, network, mount, and user namespace to isolate each container environment from the host operation system. In order to control hardware resource such as CPU, memory, network, and disk I/O, container uses cgroups to limit each of the containers resources. Container doesn't have hypervisor to isolate from the host operating system, therefore, it can offer better performance than virtual machine. There are already many software to control container, such as LXC [4], LXD, Open VZ. For now, Docker [2] is the most popular container engine. The Virtual Machine and Container architecture is shown as Fig 1.1.

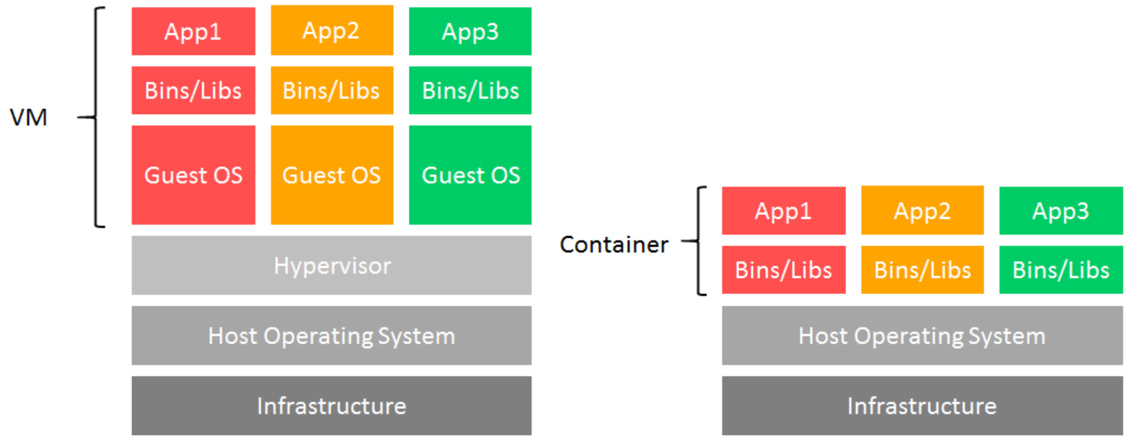



Figure 1.1: Virtual Machine and Container architecture

Checkpoint and restoration can freeze a running process state and save process information to checkpoint images. It can be used to checkpoint image files to restore the process state. These features can be used to checkpoint and restore containers because each container is a process in the host operating system.

In this paper, we choose container but not Virtual Machine to do migration between two host machines, because container need less disk storage and less network transport resources. We not only use checkpoint and restoration to migrate the containers between two physical computers but also migrate the containers in the Docker Swarm cluster. There are many Docker cluster software, like Docker Swarm, Google Kubernetes, Apache Mesos, etc. We choose Docker Swarm because it supports native Docker API, and Docker usage. We don't need to learn the other Docker cluster software. Moreover, we improve high availability and rescheduling feature in Docker Swarm using checkpoint and restoration than keep versions of container checkpoint images in the remote storage server that whenever the computer fails, Docker Swarm manager will restore the containers from the checkpoint images.

# Chapter 2

# Background

## 2.1 Docker

Docker [2] is an open-source project container engine. It provides an additional layer of abstraction and automation of operating-system-level virtualization in Linux. Moreover, Docker has extra image management and layered file system to optimize the performance of container and reduce disk space. Docker images and Docker containers share the same image layer file if they have the same data. When Docker creates container, Docker will create a writable layer that allow the container application to change file system. Docker has two parts; Docker Client and Docker Daemon.

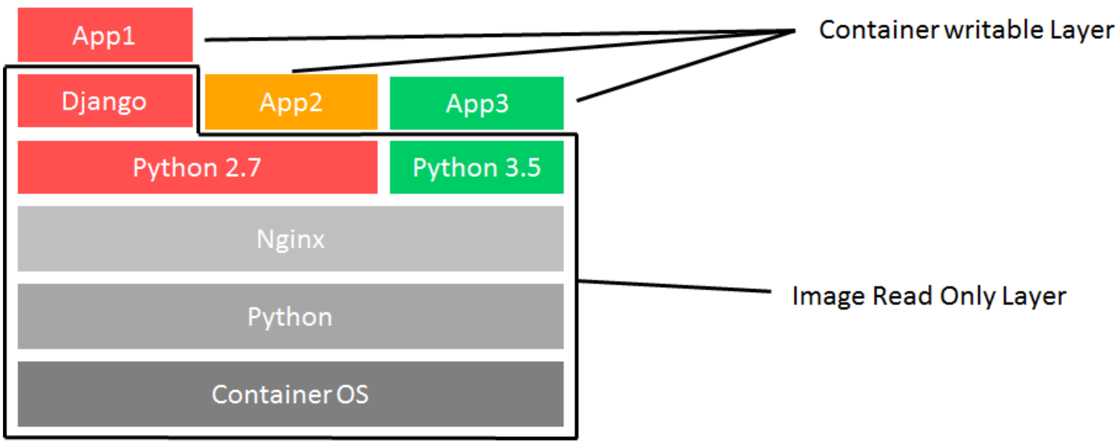



Figure 2.1: Docker layered file system

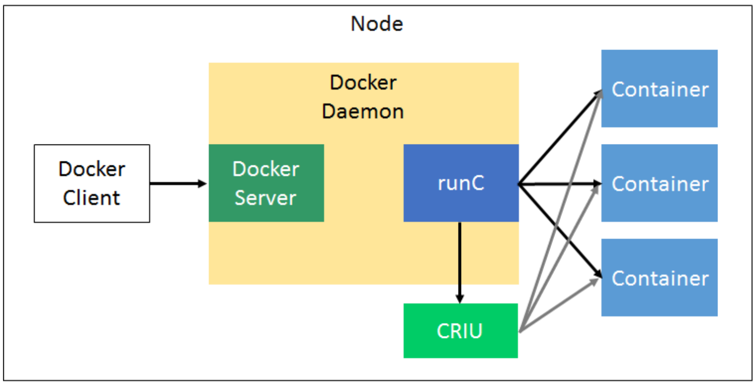



Figure 2.2: Single node Docker

### 2.1.1 Docker Client

Docker is typical client/server architecture application. Docker Client is a CLI (Client Line Interface) in Docker. Docker uses Docker Client to send and receive requests to Docker Daemon. Also, Docker supports remote RESTful API to send and receive HTTP requests to Docker Daemon. In additional, it has been implemented by more than 10 programming languages.

### 2.1.2 Docker Daemon

Docker Daemon is a daemon that runs as system service. It has three of the most importance features:

- Receive and handle requests from Docker Clients.
- Manage containers.
- Manage container images.

When Docker Daemon is running, it will run a server that receives requests from Docker Clients or remote RESTful API. After receiving requests, server will pass the requests by router to find the handler to handle the requests. By default, Docker Daemon listens to UNIX socket requests, it serves root permission, or docker group membership. Whenever user wants Docker Daemon to listen to remote RESTful API or Docker Swarm requests, it has to enable the TCP socket.

### 2.1.3 runC

When Docker came into being, Docker used LXC as its container execution environment. After Docker version 0.9, Docker dropped LXC but used libcontainer as its default execution environment. runC is basically a repackaging of libcontainer, which is a CLI tool for running containers according to the OCI(Open Container Initiative) specification. It doesn't need any dependency from the operating system, it can control the Linux Kernel, including namespace, cgroups, apparmor, network, capabilities, firewall, etc. runC provides a standard interface to support the containers management that Docker can use it to control the containers.

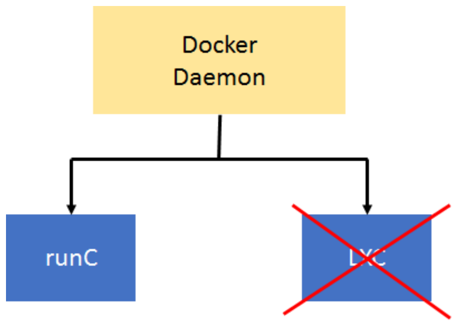



Figure 2.3: Docker Daemon execution environment with runC and LXC



## 2.2 CRIU

CRIU [1] (Checkpoint/Restore in Userspace) stands for Checkpoint and Restore in User Space, creates a complete snapshot of the state of a process, including things like memory contents, file descriptors, and even open TCP connections. It can be used for suspending and resuming processes, or migrating them from one machine to another.

### 2.2.1 Checkpoint

**Pre-dump**

**Checkpoint**

**Restore**

## 2.3 Docker Swarm

Docker Swarm [3] is a native clustering for Docker. It gathers several Docker engines together into one virtual Docker engine. Docker Swarm serves standard Docker API, so it can be connected by Dokku, Docker Machine, Docker Compose, Jenkins, DockerUI, Drone, etc. Of course, it also supports Docker Client as well.

In Docker Swarm, it has two components which are Swarm Manager and Swarm Node. Swarm Manager is the manager which handles Docker Client and RESTful API requests and manages multiple Docker Nodes' resources. Docker Node is an agent which sends heartbeat to Discovery Service to ensure Docker Daemon is alive in the cluster.

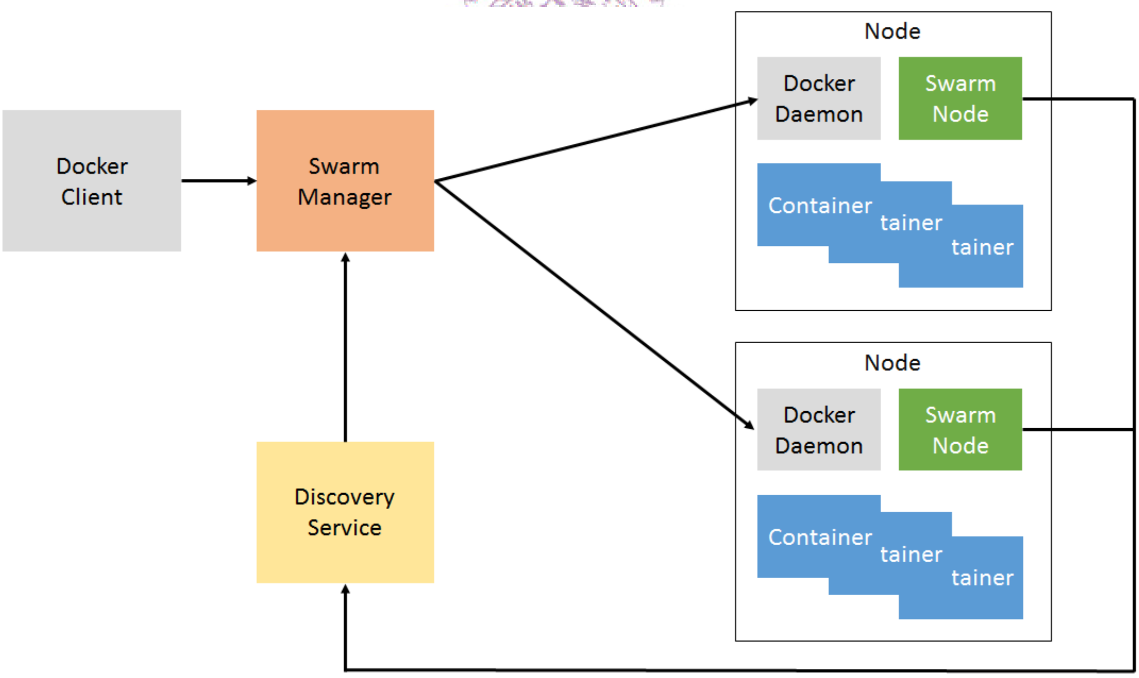



Figure 2.4: Docker Swarm architecture

### 2.3.1 Discovery services

Docker Swarm provides multiple Discovery Services backends. They are used to discover the nodes in the cluster. There are:

- Using a distributed key/value store, like Consul, Etcd and Zookeeper.
- A static file or list of nodes.
- Docker Hub as a hosted discovery service.

Otherwise, it also supports any modules that satisfy Discovery API interface.

### 2.3.2 Scheduler

Docker Swarm scheduler decides which nodes to use when creating and running a container. It has two steps: First, the scheduler follows user's filters to decide which nodes conform to it. Secondly, it undergoes multiple strategies to select the best node in the cluster.

#### Filter

Filters are divided into two categories; Container configuration filters operate on characteristics of containers, or on the availability of images on a host. Node filters operate on characteristics of the Docker host or on the configuration of the Docker Daemon.

The container configuration filters are:

- Affinity
- Dependency
- Port filter

The node filters are:

- Constraint
- Container slots
- Health filter

**Strategies**

The Docker Swarm scheduler features multiple strategies for ranking nodes. Docker Swarm currently supports these values:

- Spread
- Binpack
- Random

Spread and Binpack strategies compute rank according to a nodes available CPU, its RAM, and the number of containers it has. It selects a node at random. Under the Spread strategy, Swarm chooses the node with the least number of containers. The Binpack strategy chooses the node which has executed most containers. The Random strategy uses no computation and chooses nodes at random regardless of their available CPU or RAM.

### 2.3.3 High availability of Swarm Manager

In Docker Swarm, Swarm Manager responds to the cluster and manages the resources of multiple Docker Nodes at scale. If Swarm Manager dies, we have to create a new one and deal with the interruption of service.

The High availability feature allows Docker Swarm has multiple Swarm Manager instances. We can create a primary manager and multiple replica instances. Whenever we send requests to replica instances, it will be automatically proxied to the primary manager. In addition, if the primary manager fails, the other replica instances will lead a new primary manager.

### 2.3.4 High availability of Docker Swarm containers

In Docker Swarm, it has a rescheduling policy. As we set the reschedule policy when we start a container, whenever Swarm nodes fail, Swarm Manager will restart all of the containers to another alive Swarm Nodes.

# Chapter 3

# Design and Implementation

## 3.1 Docker

In native Docker, It has two part, Docker Client and Docker Daemon. Docker Daemon has many components include server, engine, registry, graph, driver and runC. To support dump checkpoint and restore request, some of these steps should be implemented.

### 3.1.1 Docker Client

We implement 3 Docker commands in Docker Client, including checkpoint, restore, migrate. In checkpoint command should have these configurations:

- image directory - Dump checkpoint image directory.
- work directory - Dump checkpoint image log directory.
- leave running - After dumping checkpoint image, keeping running container or not.
- pre-dump - Pre-dump checkpoint memory image to minimize frozen time.
- pre image directory - Define which version image to compare.
- track memory - Track memory to pre image directory image to minimize disk space.

In restore command should have these configurations:

- image directory - Checkpoint image directory to restore from.
- work directory - Directory for restore log.
- force - Fore restoring container from image directory whether container is running or not.

In migrate command, it focus on Docker Swarm Scheduler filter configurations. In run command, we can do this with the environment variable or the label. Therefore, we implement environment variable and the label configurations in migrate command.

### 3.1.2 Docker Daemon

In native Docker Daemon, it doesn't support checkpoint and restore command. Fortunately, it is already implemented in runC, so we have to add a proxy between Docker Daemon and runC, which can handle Docker Client's checkpoint and restore requests.

## 3.2 Docker Swarm configuration

As Figure 3.1, we prepare a remote storage server for saving Docker containers dump checkpoint images. It should have fault tolerant to avoid service shout down. For these reasons, We choose glusterFS to be our experiments remote storage server.

## 3.3 Docker containers migration in Docker Swarm

Docker Swarm creates containers through Swarm scheduler to dispatch Docker nodes. If we want to specific assign which node we want to create containers, we have to set filters like constraint, affinity or dependency. To migrate containers in Docker Swarm, we must avoid the containers which we want to migrate that migrate to an other nodes, instead of the same node.

Step 1. Check Docker Swarm cluster has at least two Swarm nodes.

Step 2. Parse Docker Client requests to analyse label and environment variables, and transform label and environment variables to Docker Swarm filters.

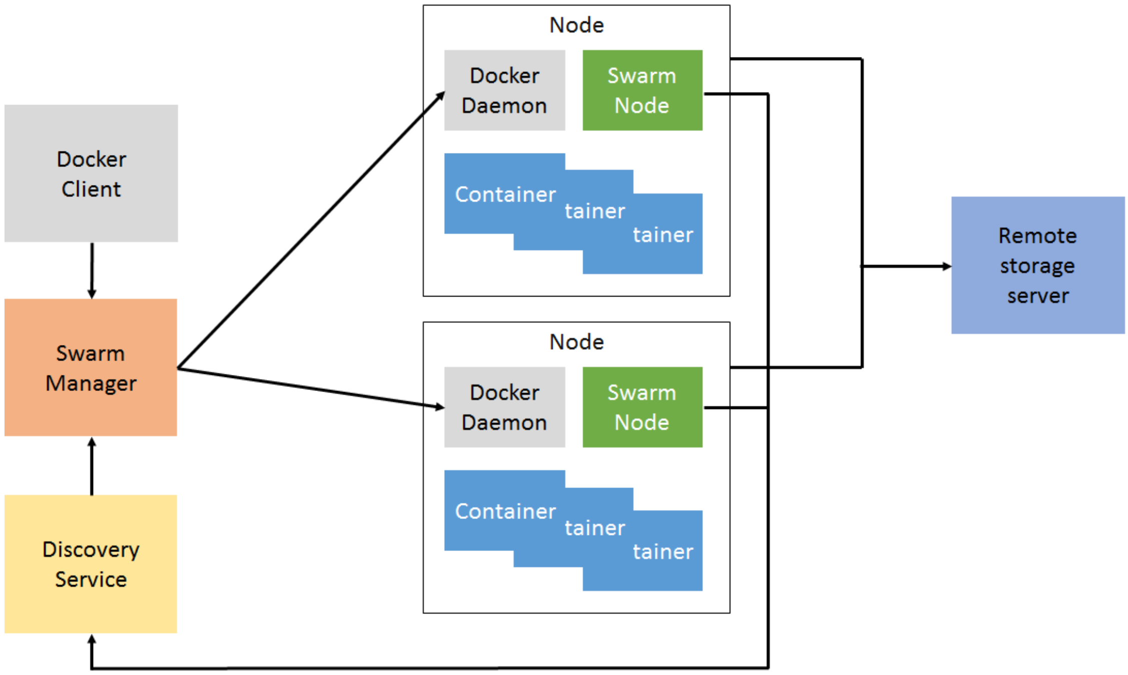



Figure 3.1: Docker Swarm with remote storage server

Step 3. Add constraint filter to make sure the container which we want to migrate does not migrate to the same node.

Step 4. Create empty container on the Docker Swarm scheduler chooses node, this step could download the container image if it doesn't exist in the be chosen node.

Step 5. Pre-dump the container checkpoint image which we want to migrate to decrease container frozen time.

Step 6. Dump the container checkpoint image by tracking memory from pre-dump checkpoint image.

Step 7. Restore the container to the empty container that Docker Swarm scheduler chooses node.

Step 8. Delete the checkpoint images.

Step 9. If the container was migrated which had set checkpoint restore rescheduling policy, it will restart checkpoint restore rescheduling policy 3.4.

## 3.4 Docker Swarm checkpoint and restoration rescheduling policy

In Docker Swarm, it has rescheduling policy. As we set the reschedule policy when we start a container, whenever Swarm nodes fail, Swarm Manager will restart the containers which on the fail nodes to another alive Swarm Nodes.

We improve this policy that we dump the checkpoint image for every containers which we want to keep the container checkpoint for every containers checkpoint ticker pounding. Whenever Swarm Nodes fail, Swarm Manager will restore the containers which Swarm Manager has dumped the checkpoint. Otherwise, the checkpoint tickers policy provides version of checkpoint image by tracking memory. It only dump different memory page checkpoint to new version checkpoint image.

In addition, it also support high availability that whenever Docker Swarm primary manager fails, the others Swarm Manager replica instances will lead a new primary manager. After replica leading a new primary manager, it will restart container checkpoint tickers.

### 3.4.1 Docker Swarm Container Checkpoint Tickers Strategy

The details of our algorithm in Algorithm 1 that explains Container Checkpoint Tickers Strategy. In line1 and line2, we set checkpoint-time $T_i$ and version-group $VG_i$ when we create the container $C_i$ through Docker Swarm Manager, and every $T_i$ are independent. Each $T_i$ in the Docker Swarm cluster has its starting time and time ticker, time ticker will dump checkpoint image repeatedly at regular intervals. If version-group $VG_i$ doesn't be set, it will be set to 5. We define checkpoint version $V_i = 0$, and pre-dump checkpoint version $\textit{pre-dump}V_i = 0$ for each container $C_i$.

In line4-18, for each container is running and $C_i$ checkpoint ticker $T_i$ is time up, $C_i$, Swarm Manger will do line5-17. In line6-9, if the container $C_i$ doesn't has any pre-dump image or newest version-group $VG_i$ checkpoint is full, Swarm Manger will send the request to Swarm Node $N_i$ to create a new directory and pre-dump the checkpoint image as pre-dump version $\textit{pre-dump}V_i$. After pre-dumping the container checkpoint image or $C_i$ has previous version checkpoint images, Swarm Manager sends request to dump checkpoint image to Swarm Node $N_i$ to dump checkpoint version $V_i$. Whenever dumping checkpoint image, CRIU will track memory different to the previous checkpoint image or the pre-dumping checkpoint image do reduce image dick space. As is shown as Figure 3.2, every container has each pre-dump checkpoint directories and each pre-dump checkpoint directories has version-group versions checkpoint images. In line10-12, Swarm Manager Sends delete checkpoint request to $N_i$ to delete oldest Pre-dump version directory whenever container has more than three Pre-dump versions directory.

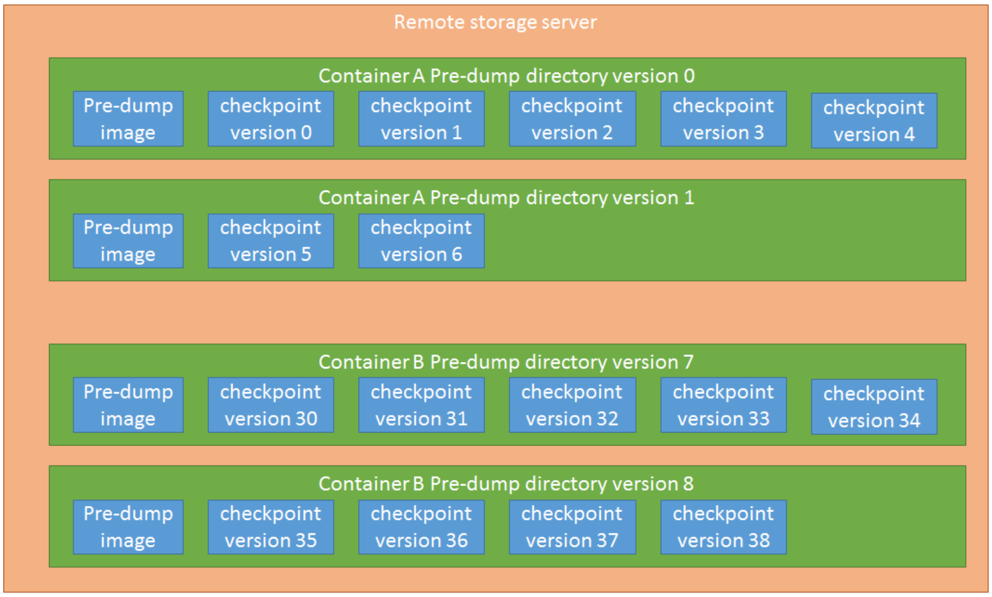



Figure 3.2: Containers checkpoint versions in remote storage server

### 3.4.2 Docker Swarm Restore Rescheduling Policy

Algorithm 2 gives a detailed explanation of restore Rescheduling Policy. In line1, we set $R_i$ label when we create the container $C_i$ through Docker Swarm Manager. In line3-7, Whenever Swarm Nodes $N_i$ fail, Swarm Manager will execute procedure RESTORE CONTAINER that restore every containers $C_i$ which has $R_i$ label to another Swarm Nodes $N_i$.

In line9-21, each container $C_i$ executes procedure RESTORE CONTAINER, Swarm Manager will choose a Swarm Node $N_r$ and create an empty container $C_r$ in $N_r$. In this step, Swarm Manager will check $N_r$ has container image or not, if $N_r$ doesn't

**Algorithm 1** Checkpoint ticker algorithm

**Input:** The set of Swarm Nodes: $\{N_1, N_2, ..., N_m\}$; The set of containers: $\{C_1, C_2, ..., C_n\}$; Version-group: $VG_i$; Checkpoint-ticker time: $T_i$;

**Output:** pre-dump images *pre-dump*$V_i$; dump images $V_i$ ;

Set $T_i$ and $VG_i$ labels when create the container $C_i$;
initial $V_i = 0$, *pre-dump*$V_i = 0$;

**for** each $C_i$ in $N_i$ **do**
    **while** $C_i$ running & $T_i$ pounding **do**
        **if** version % $VG_i == 0$ **then**
            pre-dump checkpoint image *pre-dump*$V_i$;
        **end if**
        dump checkpoint image $V_i$;
        **if** $VG_i$ directory $> 3$ **then**
            delete oldest pre-dump checkpoint *pre-dump*$V_i$ directory;
        **end if**
        $V_i = V_i + 1$;
        **if** $V_i$ % $VG_i == 0$ **then**
            *pre-dump*$V_i$ = *pre-dump*$V_i + 1$;
        **end if**
    **end while**
**end for**

have container image, $N_r$ will download the container image from Docker remote registry and create the empty container $C_r$. As a result of creating container $C_r$, Swarm Manager restores the fail container $C_i$ checkpoint image to $C_r$. To avoid restoring fail, Swarm Manager will retry second last version checkpoint to restore, and it will retry version-group(default is 5) times. The container $C_i$ checkpoint image will be delete after restoring the container $C_i$.

If Swarm Manager retries version-group all fail, it will create and start a new container as normal Docker Swarm rescheduling policy.

### 3.4.3 High availability of Swarm Manager in Docker Swarm checkpoint and restoration rescheduling policy

Whenever Docker Swarm primary manager fails, the others Swarm Manager replica instances will lead a new primary manager. After replica leading a new primary manager, it searching every Docker Node's containers which has checkpoint restore rescheduling policy's label. If the containers has checkpoint restore rescheduling policy's label, Docker Swarm new primary manager will restart container checkpoint tickers.

**Algorithm 2** Restore rescheduling policy algorithm

**Input:** The set of Swarm Nodes: $\{N_1, N_2, ..., N_m\}$; The set of containers: $\{C_1, C_2, ..., C_n\}$; Version-group: $VG_i$; Checkpoint-ticker time: $T_i$; Restore rescheduling policy label:$R_i$;

**Output:** Restored containers:$C_{ri}$;

Set $R_i$ labels create the container;

**if** $N_i$ fail **then**<br>
    **for all** $C_i$ in fail $N_i$ has $R_i$ **do**<br>
        Restore Container;<br>
    **end for**<br>
**end if**

**procedure** Restore Container<br>
    $N_r \longleftarrow$Create a empty container $C_r$<br>
    **for** $V_i$ downto $V_i$ - $VG_i$ **do**<br>
        $C_{ri} \longleftarrow$ Restore checkpoint $V_i$;<br>
        **if** Restore $C_{ri}$ success **then**<br>
            **break**<br>
        **end if**<br>
    **end for**<br>
    Delete the container checkpoint image;<br>
    **if** Restore container $C_i$ fail **then**<br>
        $C_r \longleftarrow$ Restart the container; $C_i$;<br>
    **end if**<br>
**end procedure**

# Chapter 4

# Experiments

## 4.1 Environment

We developed our experiments environment cluster were listed as table 4.1, and created 4 virtual machines in the computer that each of their resource were listed as table 4.2. We mount NFS to each virtual machines, and test 2 type of NFS, hard disk, and memory(tmpfs). In addition, we test difference about dumping new checkpoint image each time and pre-dumping checkpoint image than dumping checkpoint follow pre-dump checkpoint image.

| | |
|---|---|
| CPU | Intel Xeon(R) CPU-E5-2630 v3 @ 2.40GHx x 32 |
| Memory | 64 GB |
| Disk | 1.0T |
| OS | Ubuntu 15.10 64 bit |
| Kernel version | 4.4.4-040404-generic |
| Docker version | 1.10.3 |
| Docker Swarm version | 1.2 |
| Virtual Box version | 5.10.14 |

Table 4.1: Experiment environment

| | |
|---|---|
| CPU | Intel Xeon(R) CPU-E5-2630 v3 @ 2.40GHx x 4 |
| Memory | 8 GB |
| Disk | 40G |
| OS | Ubuntu 15.10 64 bit |
| Kernel version | 4.4.4-040404-generic |
| Docker version | 1.10.3 |
| Docker Swarm version | 1.2 |

Table 4.2: Virtual Machine experiment environment

## 4.2 Container Migration Time

As shown in Figure 4.1, native is a reference line of the other kinds of experiment. We choose redis[7] as our migration container, because it uses memory to save its data. Therefore, we can check the correctness of redis data to make sure container migration is success.

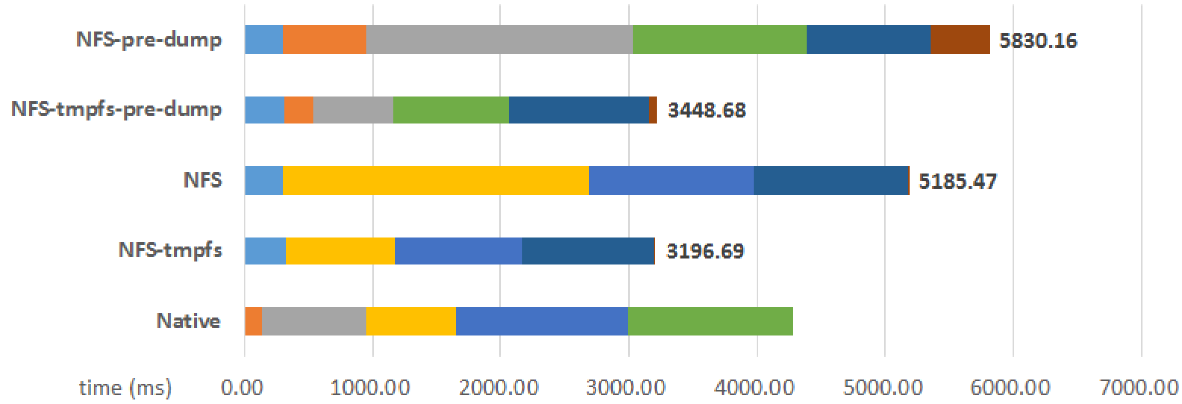
The first step of Creating the container on another Swarm Node is fast for all environments, because we assume that they already have downloaded their container images from the remote repository, thus, they don't need to transport container images from the other Swarm Node.

Second, if checkpoint ticker has version control of checkpoint the container, the container has to pre-dump the checkpoint image. After pre-dumping the checkpoint image, dump checkpoint will track memory different with the pre-dump checkpoint image. On the other hand, checkpoint ticker dump the checkpoint image directly. The result of total checkpoint time, the version which has pre-dumping one's time is longer then the other one, because it has to add pre-dumping checkpoint time and dumping checkpoint time. Although the version which has pre-dumping one's is longer, but it provides the less frozen time to the container that improves CPU's utilization.

The third step is restoring the container to the container which has already been created in the Swarm Node. except NFS with memory, they all have nearly performance.

The last step, it has big disparity of the delete checkpoint image step at pre-dumping version, because it has more image files and directories than the version without pre-dumping version.

After all, the checkpoint and restoration of container in memory is more faster then hard disk in NFS. As result of Figure 4.1, NFS with memory has the best performance in this experiment.




| | Native | NFS-tmpfs | NFS | NFS-tmpfs-pre-dump | NFS-pre-dump |
|---|---|---|---|---|---|
| Create container | | 319.20 | 300.80 | 310.00 | 295.55 |
| Pre-dump | 133.88 | | | 225.08 | 657.86 |
| Checkpoint-pre-dump | 813.36 | | | 631.95 | 2080.38 |
| Checkpoint | 699.32 | 858.41 | 2385.28 | | |
| Restore-pre-dump | 1346.70 | 992.76 | 1294.80 | | |
| Restore | 1294.80 | | | 891.65 | 1358.67 |
| Remove container | | 1020.29 | 1198.54 | 1099.38 | 959.44 |
| Delete checkpoint | | 5.44 | 5.43 | 59.43 | 467.28 |

Create container  Pre-dump  Checkpoint-pre-dump
Checkpoint  Restore-pre-dump  Restore
Remove container  Delete checkpoint

Figure 4.1: Docker Swarm migration with remote storage server

## 4.3 Container Checkpoint Time Influence of Container Process Time

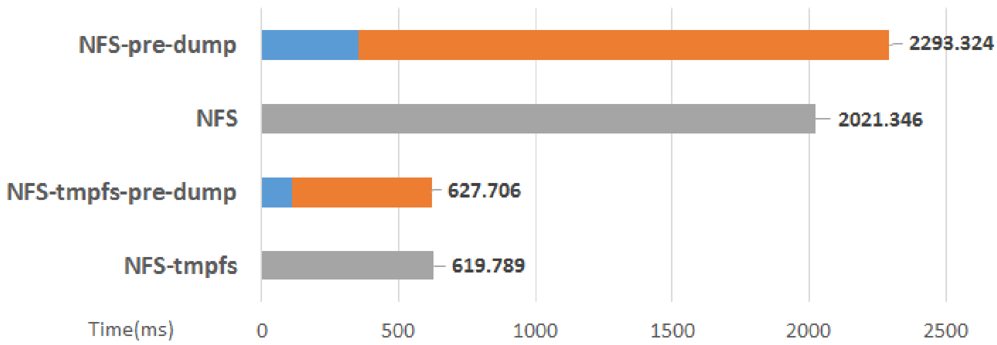
In this experiment, We use sysbench[6] to test performance of process's CPU execution time. Our parameter of sysbench is:

sysbench –test=cpu –cpu-max-prime=20000 run

It will run around 30 seconds in native container without dumping any checkpoint.

In Figure 4.2, this figure lists every checkpoint time in the experiment environments, also, if remote server uses the memory to save the checkpoint images, it will got better performance. In the checkpoint restore rescheduling policy (section 3.4), we set a parameter of checkpoint-ticker period $T_i$ which will checkpoint repeatedly for every checkpoint-ticker timing up. As Figure 4.3, the result of the checkpoint-ticker period $T_i$ is in direct ratio to the container process execution time. As observe the result, if the remote service doesn't use memory, and the checkpoint-ticker period $T_i$ is smaller than 5, container will take a half time at dumping checkpoint image. If the checkpoint-ticker period $T_i$ is smaller than 3, container even will take over twice times larger than the native container process execution time.

As these results, the checkpoint-ticker period $T_i$ has a big influence of container execution time, it is a important parameter in checkpoint restore rescheduling policy that if $T_i$ is too small, container will get a bad performance, but if $T_i$ is too big, whenever Swarm Node fails, the restore container needs to execute the process again. The worst, it might lost some important data.

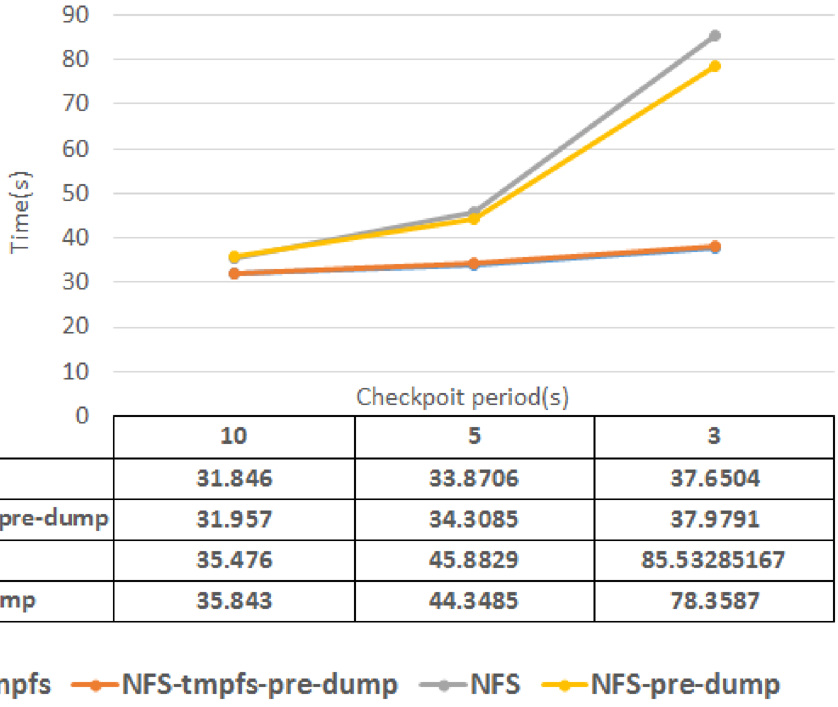



| | NFS-tmpfs | NFS-tmpfs-pre-dump | NFS | NFS-pre-dump |
|---|---|---|---|---|
| Pre-dump | | 107.718 | | 350.33 |
| Checkpoint-pre-dump | | 512.071 | | 1942.994 |
| Checkpoint | 627.706 | | 2021.346 | |

Pre-dump Checkpoint-pre-dump Checkpoint

Figure 4.2: Container Checkpoint Time of Container Process Time

## 4.4 Container Memory Size Influence of Container Checkpoint Time




| | 10 | 5 | 3 |
|---|---|---|---|
| NFS-tmpfs | 31.846 | 33.8706 | 37.6504 |
| NFS-tmpfs-pre-dump | 31.957 | 34.3085 | 37.9791 |
| NFS | 35.476 | 45.8829 | 85.53285167 |
| NFS-pre-dump | 35.843 | 44.3485 | 78.3587 |

Figure 4.3: Container Checkpoint Time Influence of Container Process Time

# Chapter 5

# Related Work

[5]

# Chapter 6

# Conclusion

CRIU provides the container for dumping the checkpoint image and restoring it in the single host machine. In this thesis, we have purposed it to expand to Docker Swarm in the cluster. To support high availability in the Docker Swarm cluster, we also use checkpoint ticker and restoration to recover last container work state in the other Swarm Node.

In the future, we plan to live migration the Docker container to support network applications' migration. In addition, we want to extend power consumption algorithm in Docker Swarm to enhance power consumption saving in the Docker Swarm cluster.

# Bibliography

[1] Criu. `https://criu.org/Main_Page`.

[2] Docker. `https://www.docker.com/`.

[3] Docker swarm. `https://docs.docker.com/swarm/overview/`.

[4] Matt Helsley. Lxc: Linux container tools. *IBM devloperWorks Technical Library*, 2009.

[5] Michael R Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60. ACM, 2009.

[6] Alexey Kopytov. Sysbench: a system performance benchmark. *URL: http://sysbench. sourceforge. net*, 2004.

[7] Matti Paksula. Persisting objects in redis key-value database. *University of Helsinki, Department of Computer Science*, 2010.