A STUDY OF A NATIVE DOCKER CLUSTERING SYSTEM ON A POWER-CONSTRAINED ENVIRONMENT

Chanwit Kaewkasi, Wichai Srisuruk, Bhuridech Sudsee, Pathawee Ngoenthai
School of Computer Engineering, Suranaree University of Technology
chanwit@sut.ac.th, wichai@sut.ac.th, m5741861@sut.ac.th, m5741847@sut.ac.th

ABSTRACT A data center is a vital component for business. A corporate usually prefers a private data center to secure its data, but it is costly to operate its own data center. The expenditure covers constructing the building, purchasing server machines and devices as well as the operating, electricity and system cooling cost. This raises the importance of applying low-powered CPUs to build a computing cloud.

A cloud platform usually employs virtualization technologies to utilize its resource sharing. Among widely used hypervisor-based virtualizations, there is another efficient technique, the OS-level virtualization. This paper focuses on a clustering technology based on Docker Swarm, hereinafter referred as *Swarm*. It is the native clustering solution developed by Docker inc. in collaboration with our laboratory and other contributors.

This paper describes the work and experimental results of studying Swarm in the context of a power-constrained environment. One of the finding is that Swarm scales linearly even on a 50-node cluster. The experimental results showed that Docker Swarm can form a cluster effectively. It is also showed that Swarm scales linearly even on a 50-node 32-bit ARMv7 cluster. The study also found that our 32-bit cluster hits the limitation of running containers at maximum 93 containers per node.

1. INTRODUCTION

A data center is a vital component for business. Beside server machines, it consists of other supporting systems, such as the redundant power supply, the cooling system as well as the security systems (Bullock, 2009). A corporate usually prefers a private data center to secure its data, but it is costly to operate its own data center. The expenditure covers constructing the building, purchasing server machines and devices as well as the operating cost. This leads to an expectation that the number of data centers will be increasing until 2017, then decreasing afterwards (Villars & Shirer, 2014) (Smolaks, 2014). One of the main reasons is the economy burden of operating a data center. Its operation and maintenance cost is fixed despite no one in the corporate uses it.

Cloud computing is a solution to solve the cost of building a corporate-owned data center. A cloud platform usually employs virtualization technologies to utilize its resource sharing (Marinescu, 2013). Among widely used hypervisor-based virtualizations, there is another efficient technique, the OS-level virtualization, to manage the cloud's resources. Docker is an implementation of such technique, which is called containerization instead of virtualization. As this technique shares the same kernel of the host machine, Docker can start a new container without spending time to boot the machine layer (Felter et al., 2014). To form and manage a cluster of Docker instances, it can be done using Docker Swarm (Vieux et al., 2014), hereinafter referred as *Swarm*. Swarm is the native clustering solution developed by the same Docker developer team with helps of individual contributors. As reported by Kaewkasi (2015), Swarm is also able form a cross-platform hybrid cluster, which utilizes both ARM and x86 hardware underneath but serve as a single manageable Docker instance. Previous works reported by Kaewkasi & Srisuruk (2014a) and Kaewkasi & Srisuruk (2014b) encountered difficulties to manage software stacks on ARM-based clusters as the constrained environment is not suitable for hypervisors. These previous works motivated us to study a Docker-based virtualization cluster, Swarm, in terms of processing overhead and scalability on an ARM-based cluster.

This paper describes the work and the experimental results on a study of Swarm in the context of a power-constrained environment. Experiments have been conducted to find out answers for the following research questions. Firstly, how large is the overhead of a single-node Swarm over Docker compared to a plain Docker instance in the low-power environment? Secondly, how does a Swarm cluster behave in term of scalability when deploying short-lived containers to it? Finally, what is the

behavior and limitation of an ARM-based Swarm cluster under steady loads? One of the main contributions of this paper is the finding that Swarm scales linearly even on a 50-node cluster.

The remainings of this paper are organized as follows. Section 2 discusses related works. Section 3 reviews Swarm, the native clustering system for Docker. Section 4 discusses the technical configuration of the system prepared for the experiments. Section 5 discusses the experiments and their results. This paper ends with Section 6 for discussion, conclusion and future works.

2. RELATED WORKS

As Swarm is still being developed, there is no scientific study on a Swarm cluster yet. This section reviews works related to Docker-based clusters in general.

Liu and Zhao (2014) were trying to create a cloud computing system using two high-end machines. They created 6 containers on each of them. Each container open ports ranging from 49153-65535 to allow data communication with the host. They have found that containers performed faster than virtual machines with the same applications and settings. Our work goes further by studying Swarm to form a Docker cluster.

Williams (2015) designed a cluster architecture using heterogeneous hardware. The author used an ARM and another x86 machine to form a Swarm cluster. But he employed Powerstrip (Marsden, 2015), an API prototyping tool for Docker, to intercept calls from Swarm, so the architecture is incurred from extra overheads generated by Powerstrip.

Falck (2014) developed an ARM cluster with 4 boards of Raspberry Pi and installed Docker on them. The author used SaltStack to manage the cluster. However there was no study of the performance of SaltStack. The work described in this paper used Swarm to form the cluster and studied its characteristics.

Recently, Kaewkasi (2015) has created a Cross-platform Hybrid Cloud using Swarm and Docker. The cluster was formed atop a 50-node Aiyara cluster (ARM) and another 50 DigitalOcean nodes. Swarm was tweaked to make it able to place a certain container image to a right platform. In his work, there was no performance measurement of the cluster yet. Differ from Kaewkasi (2015), the work described here in this paper measured Swarm characteristics and used the same ARM hardware but excluded the DigitalOcean nodes.

Google has also developed Kubernetes (Bernstein, 2014) to support clustering Docker instances. However, there is no performance study of Kubernetes available yet.

3. DOCKER SWARM

Swarm (Vieux et al., 2014) is a clustering system natively built for Docker. It has been developed using an open process, which allows one of the authors to join the development of the project since the beginning. To make this paper self-contained, the structure of Swarm is described here in this section. A Docker instance running on a Node is called an Engine. In this paper, we use the terms *Docker Engine* and *Docker instance* interchangeably.

There is an assumption that each machine (Node) will start only one Engine. There is a *SwarmAgent* running via the `swarm join` command on each Node. A SwarmAgent is responsible for advertising the address of the Engine on its Node by registering the address to a *Discovery* service. A Discovery service is a store that keeps a list of healthy Engines. The *SwarmManager* will retrieve the list of Engines to form the cluster as a virtual Docker host. Any *DockerClient* can just interact with this virtual host as it is another Docker Engine.

SwarmManager is responsible for selecting a certain Engine to place a newly created container. During this process, it applies a set of *Filters* to available Engines so that an Engine is chosen for conditions specifying as *Constraint* or *Affinity* expressions.

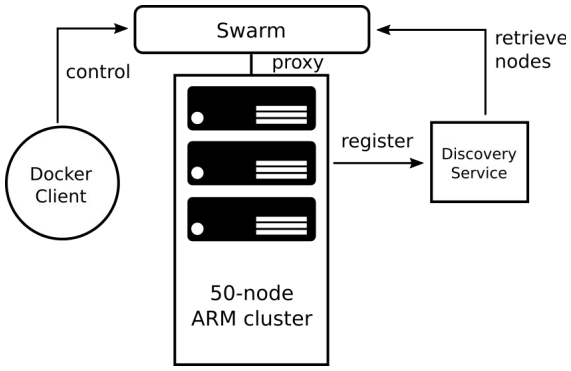

Figure 1. A block diagram of cluster configuration

4. CLUSTER CONFIGURATIONS

Cluster configurations based on Swarm are discussed in this section. Figure 1. shows the block diagram illustrating Swarm acting as a single virtual endpoint for 50 instances of Docker running on the physical cluster. Each Docker instance is registered to the central *Discovery Service*, from where *Swarm* retrieves the list of Docker nodes. A Docker client then will be able to talk to Swarm because it now serves the similar interface as a normal Docker host. Figure 2. Block diagram illustrating the 50-node physical cluster proxied by Swarm.

The Docker client runs on a 3.2 GHz x86 64-bit Linux machine with 4GB RAM. We installed the official Docker engine version 1.6 on it. On this 50-node ARM cluster, we use the Aiyara cluster model (Kaewkasi & Srisuruk, 2014a), which is built atop a set of ARM Cortex-A7 dual-core 1 GHz processors. The Docker engine running on the ARM cluster is a custom build based on Docker 1.6.

```
int main() {
    struct timespec time1;
    time1.tv_nsec = 500000000L;
    return nanosleep(&time1, null);
}
```

Figure 2. Cluster payload with *nanosleep* for 500 ms.

```
FROM        nikitav/busybox-arm
ENV         LD_LIBRARY_PATH /lib
COPY        nanosleep /
ENTRYPOINT /nanosleep
```

Figure 3. Cluster payload *nanosleep* for 500 ms.

In the experiments, a payload for the ARM cluster has been prepared as a program that sleep for exactly 500 ms using the system call *nanosleep* as described in Figure 2. After preparing the *nanosleep* executable, a Docker image containing the executable was prepared using the following `Dockerfile`. The Docker image is built and named `aiyara/nanosleep500ms`, as described in Figure 3.

5. EXPERIMENTS

This section describes three experiments in corresponding to three research questions. The questions are restated as the following. *RQ1*: how large is the overhead of a single-node Swarm over Docker compared to a plain Docker instance in the low-power environment? *RQ2*: how does a Swarm cluster behave in term of scalability when deploying short-lived containers to it? *RQ3*: what is the behavior and limitation of an ARM-based Swarm cluster under steady loads?

5.1 Overhead on a single node

The first experiment was conducted to answer *RQ1*: how large is the overhead of a single-node Swarm over Docker compared to a plain Docker instance in the low-power environment?

To measure a plain Docker, the experiment started by installing the Docker engine on an ARM board. Then started a payload container, `aiyara/nanosleep`. The container runtime is measure using the time command:

```
$ time `docker run aiyara/nanosleep`
```

The container was deleted by a separate command, `docker rm`, after finish. The experiment was repeated 20 times.

To measure a single-node Swarm, a Docker engine was provisioned using Docker Machine with our Aiyara driver. Docker is setup this way because it is a method suggested by the Docker orchestration workflow. Swarm retrieves the node address via an instance of ZooKeeper, a discovery service. The ZooKeeper discovery service was chosen because it is a development done by our team for the Docker Swarm project.

The ZooKeeper instance served at TCP port 2181.

The Swarm manager was started on the x86 machine. Then the Docker client connected to the Docker Engines via Swarm, rather than directly connected to each Engine. The same payload container was started through Swarm. This step was repeated 20 times.

Figure 4 shows the result from the experiment 1. The result showed that controlling containers via Swarm averagely took 2,507 milliseconds with insignificant standard derivative. In the case of controlling the plain Docker without Swarm, it averagely took 1,333 milliseconds. It is obvious that the overhead caused by Swarm is around 1.88 times of the using plain Docker as Swarm introduced another layer of processing via the network.

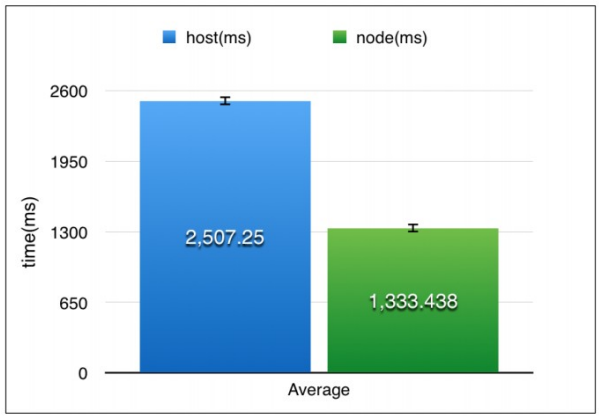

Figure 4. The experimental result of a single-node Swarm compared to a Docker instance.

5.2 Overhead when running short-live containers overhead on a single node

The second experiment was conducted to answer *RQ2*: how does a Swarm cluster behave in term of scalability when deploying short-lived containers to it? A short-live container means the container will be run for a short period of time, in this case 500ms. Then the container will be left there inside Docker with the `Exited` state. There are 5 rounds in this experiment, number of nodes were scaled gradually from 10 to 50. On each round, 625 containers were created and run spreadly across Docker Engines. We chose this number of containers to make the cluster imbalance during the experiments. Tests were repeated 5 times.

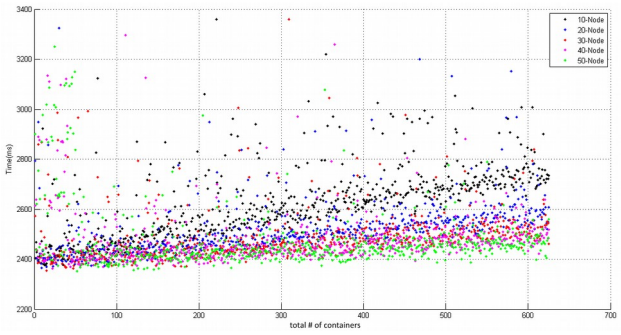

Figure 5. Time distribution used by running short-live containers in the experiment 2.

The results from this experiment are shown in Figure 5 and 6. In Figure 5, it presents the distribution of running time for each container in dots. It is clearly that running times went high at the beginning of the experiment because the joining process by each SwarmAgent. According to Figure 6, if the number of containers per node increases Swarm will take more time managing them. Also, if the number of nodes in the

cluster increases while the number of containers is a constant, the running time and overhead of Swarm will decrease. Table 1. shows number of containers per node and the average runtime of each container.

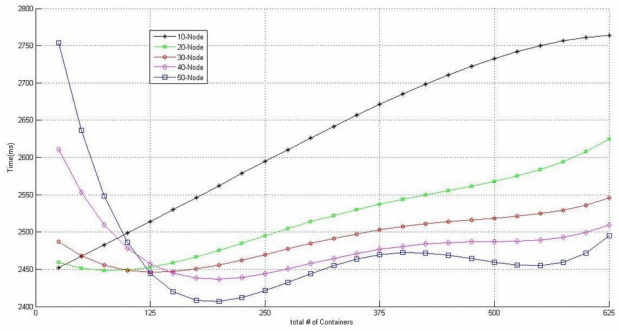

Figure 6. Average time of running short-live containers in the experiment 2.

Table 1. The result of the experiment 2 showing average number of containers per node and the average run time of each container.

# Nodes	# Containers per node	Average run time per container (millisecond)
10	62.50	2,630.31
20	31.25	2,520.64
30	20.84	2,490.76
40	15.63	2,481.49
50	12.50	2,475.13

5.3 Scalability under steady loads

The third experiment was to answer the third research question *RQ3*: what is the behavior and limitation of an ARM-based Swarm cluster under steady loads?

The setup for this experiment was similar to that of the experiment 2, but changed from running short-live containers to long-live containers and focused on the maximum number of containers on each node. Instead of measuring runtime, we measured starting time in this experiment.

The payload used in each long-live container is the `sh` command running in background. This means that each container would be in the memory forever until manually removed. The experiment started by forming a 10-node cluster, each of them gradually started containers until its number reached 90. The main reason the number of containers per node is capped to this number because the hardware of a node is 32-bit. If the number of containers goes beyond 93, a node will fail because the default stack allocation of `pthread` is at 13 MB and it makes the system ran out of memory.

The result of this experiment is shown in Table 2 and illustrated in Figure 7. From the result, it is found that adding nodes to the cluster significantly reduced stress of the cluster when starting new containers. Spikes at the beginning of the graph, the red wall, caused by each SwarmAgent joining the cluster. There was some spikes during the testing of the 30-node configuration too. It was caused by a networking problem during the experiment.

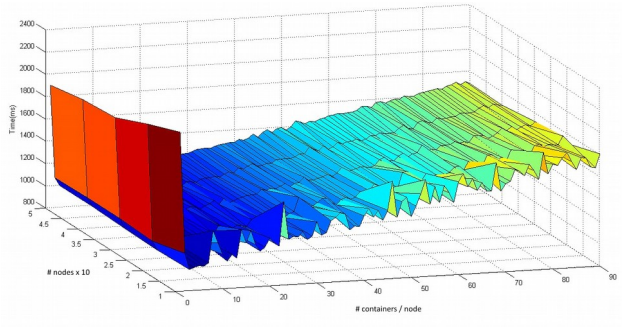

Figure 7. The graph illustrating the scalability of nodes on a Swarm cluster at 90 containers per node.

Table 2. The summary of the container starting time at 90 containers per node in average.

# Nodes	Average starting time of the a container (milliseconds)
10	1,483.72
20	1,421.35
30	1,442.19
40	1,388.92
50	1,397.41

6. CONCLUSION AND FUTURE WORK

This paper presented a work of studying a clustering system for an implementation of the OS-level virtualization, Docker, on a low-power cluster. Three experiments have been conducted to find overheads and scaling characteristics of Swarm on a 50-node cluster. The results showed that Swarm scales linearly even in a constrained environment, an ARM-based cluster. The overhead, 1.17 second, of a single-node Swarm is 1.88 times compared to plain Docker on an ARM node. This overhead came from network connection and the node selection mechanisms inside Swarm.

Running short-live containers also affected the performance of the Swarm cluster. Swarm took 2.47 and 2.63 seconds when the average number of containers per node is 12.5 and 62.5 respectively.

Starting long-live containers put stress on the cluster, but increasing number of nodes significantly reduced the stress. From these three experiments, it can be concluded that Swarm scales linearly even on the power-constrained 50-node cluster. It is really interesting to investigate that how large the size of a cluster Swarm could properly handle, both in its standalone Swarm mode and the mode that uses the scheduler of Mesos (Hindman et al., 2011).

There are several open questions motivated by the work reported in this paper. The maximum number of containers under a 32-bit ARM board is around 93. To make a 32-bit board be able to use as a better

experimental platform, the default stack size of `pthread` used by Docker would be tweakable. When forming a cross-platform hybrid cloud, the power consumption characteristic of the cluster is also interesting to study. Moreover, there is high latency around the time of deploying first containers on each node. This is also an interesting issue for further investigation.

REFERENCES

D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” IEEE Cloud Computing, vol. 1, no. 3, pp. 81–84, Sep. 2014.

M. Bullock, “Data Center Definition and Solutions,” Aug. 2009. [Online]. Available: http://www.cio.com /article/2425545/datacenter/data-center-definition-and-solutions.html

M. Smolaks, “Number of data centers to decrease after 2017,” Nov. 2014. [Online]. Available: http://www.datacenterdynamics.com/appcloud/number-of-data-centers-to-decrease-after-2017/91495.fullarticle

K. Falck, “A Private Raspberry Pi Cloud with ARM Docker,” Jul. 2014. [Online]. Available: http://sc5.io/ posts/a-private-raspberrypi-cloud-with-arm-docker

W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An Updated Performance Comparison of Virtual Machines and Linux Containers,” IBM Research Division, Austin Research Laboratory, Austin, TX, Research Report RC25482 (AUS1407-001), Jul. 2014.

B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, “Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center.” in NSDI, vol. 11, 2011, pp. 22–22.

C. Kaewkasi and W. Srisuruk, “A study of big data processing constraints on a low-power Hadoop cluster,” in Computer Science and Engineering Conference (ICSEC), 2014 International, Jul. 2014a, pp. 267–272.

C. Kaewkasi and W. Srisuruk, “Optimizing performance and power consumption for an ARMbased big data cluster,” in TENCON 2014 - 2014 IEEE Region 10 Conference, Oct. 2014b, pp. 1–6.

C. Kaewkasi, “Cross-Platform Hybrid Cloud with Docker,” May 2015. [Online]. Available: http://java.dzone.com/articles/cross-platformhybrid-cloud

D. Liu and L. Zhao, “The research and implementation of cloud computing platform based on docker,” in 2014 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Dec. 2014, pp. 475–478.

D. C. Marinescu, “Chapter 5 - Cloud Resource Virtualization,” in Cloud Computing, D. C. Marinescu, Ed. Boston: Morgan Kaufmann, 2013, pp. 131–161. [Online]. Available: http://www.sciencedirect.com/ science/article/pii/B9780124046276000051

L. Marsden, “Powerstrip: prototype Docker extensions today,” Feb. 2015.[Online]. Available: https://clusterhq.com/2015/ 02/02/powerstripprototype-docker-extensions-today/

V. Vieux and et. al, “Docker Swarm: a Docker-native clustering system,” Dec. 2014. [Online]. Available: http://github.com/docker/swarm

R. L. Villars and M. Shirer, “IDC Finds Growth, Consolidation, and Changing Ownership Patterns in Worldwide Datacenter Forecast,” Nov. 2014. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS25237514

M. Williams, “Heterogenous Docker Swarms Teaser,” Apr. 2015. [Online]. Available: http://matthewkwilliams.com/index.php/2015/04/28/heterogenousdocker-swarms-teaser/