http://dx.doi.org/10.5392/IJoC.2016.12.3.029

Design of Testbed for Agile Computing of MapReduce Applications by using Docker

Yunhee Kang
Division of Information and Communication
Baekseok University, 115 Anseo Dong, Baekseok University, Cheonan Korea 330-704

ABSTRACT

Cloud computing makes extensive use of virtual machines that permit for workloads, as well as resource usage, to be isolated from one another, and a hypervisor can be used by a virtual machine to construct cloud computing infrastructure. However, the hypervisor has high resource usage when constructing virtual machines, which results in a waste of allocated resources when not activated. Docker provides a more light-weight method to obtain agile computing resources based on a container technique that handles this problem. In this study, we have chosen this specific tool due to the increasing popularity of MapReduce and cloud container technologies such as Docker. This study aims to automatically configure Twister workloads for container-driven clouds. Basically, this is the first attempt towards automatic configuration of Twister jobs on a container-based cloud platform VM for many workloads.

Key words*: Docker, Testbed, Agile Computing, MapReduce, k-means, Twister.*

1. INTRODUCTION

Cloud computing makes extensive use of virtual machines which permit workloads to be isolated from one another and for the resource usage. By using hypervisor the virtual machine based technique is mainly used for construction of cloud computing infrastructure. However it uses high resource when constructing virtual machines and results in wasting allocated resources when they are not activated. The cloud platform gives a way when startups select the platforms to deploy their development and operational environment [1], [2]

When prototyping a distributed application like MapReduce, a developer needs to ensure that the application execution corresponds to the specification while its performance is not impacted by the number of nodes or by some failure scenarios [1]-[3]. Indeed, MapReduce relies on successive computing-communication steps that, if not coordinated with care, lead to performance bottlenecks and a poor scalability. However, this technology is grounded on extremely complex platforms with which it is often difficult to understand, configure and optimize details.

DevOps is a modern software development methodology focused on bringing software developers and operations staff in close alignment; it describes the conceptual and operational merging of development and operation needs, teams and technologies. The DevOps approach seeks to meld application development and deployment into a more streamlined exercise. In DevOps, a developer continuously starts to develop, deploy and test a service in diverse test-beds. How to build cloud infrastructure to run an application is an important decision for efficiently running a cloud application. In a cloud platform isolation and resource control have traditionally been achieved through the use of virtual machine(VM)s. VMs are used extensively in cloud computing. Since virtually all cloud workloads are currently running in VMs, VM performance is a crucial component of overall cloud performance. However, it is expensive for a hypervisor to run multiple OS on a single physical machine to satisfy the isolation between each application.

Docker is one of the most efficient ways to provide more light-weight for agile computing resource based on container technique to handle this problem. For this work we have chosen this specific tool due to the increasing popularity of MapReduce and cloud container technologies such as Docker. This study aims at automatically configuring Twister workloads for container-driven clouds. Basically this is the first attempt towards automatic configuration of Twister jobs on container-based cloud platform VM for many workloads.

The paper is organized as follows: Section 2 describes the related works including brief overview of MapReduce, Twister and the Docker. We report the results of our experimental study and the observations from the result of our experiments in Section 3. Conclusions are presented in Section 4.

* *Corresponding author, Email: yhkang@bu.ac.kr*
Manuscript received May. 31, 2016; revised Jul. 06, 2016; accepted Jul. 13, 2016

2. RELATED WORKS

2.1 MapReduce

MapReduce is an emerging programming model for a data-intensive application proposed by Google, which has been widely used in industry and academia. It borrows ideas from functional programming, where a programmer defines map and reduces tasks to process a large set of distributed data. Traditional parallel applications are based on a runtime library that has some features of communication and synchronization [4]. MapReduce programs are designed to compute large volumes of data in a parallel fashion. It is a kind of data parallel languages aimed at loosely coupled computations that execute over given data sets [2]. This requires dividing the workload across a large number of machines. The degree of parallelism depends on the input data size [2]-[4].

In a MapReduce application supported by a MapReduce library, all map operations can be executed independently. Each reduce operation may depend on the outputs generated by any number of map operations. All reduce operations can also be executed independently. The following describes the MapReduce programming model:

- The computation takes a set of input (key, value) pairs, and produces a set of output (key', value') pairs. The computation is expressed as two functions: Map and Reduce.
- The Map takes an input pair and produces a set of intermediate (key, value) pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key key' and passes them to the Reduce function.
- The Reduce accepts an intermediate key key' and a set of values of that key. It merges together these values to form as small a set of values as possible.

In this programming model the computation takes a set of (key, value) pairs, and produces a set of output (key, value) pairs. It consists of two functions: map and reduce. These two functions have the following signatures:

Map :: (key1, value1)→list((key2, value2))
Reduce :: (key2, list(value2))→list((key3, value3))

Fig. 1 shows the flow of MapReduce processing, and the input data get partitioned in small blocks of a certain size for MapReduce task. Mapper for map task reads with the given input data and generates a pair of key-value that is the basic data structure for the application processing. Reducer for reduce task generates an output of all the values related to the equivalent intermediate key in a pair of key-value. During this process, the group by computation as to the intermediate value is collated and the key-value output from each Reducer is to be re-used in the distributed storage system. Mapper and Reducer process data in parallel. A MapReduce program is composed of a Map() procedure that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() procedure that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).

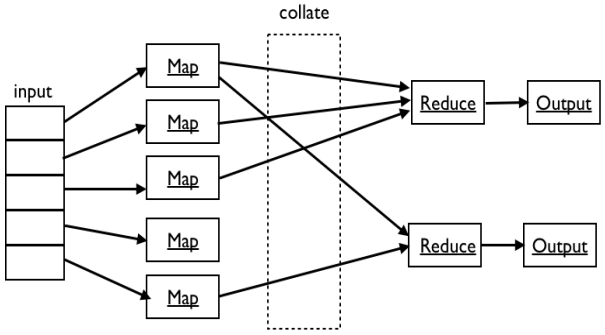

Fig. 1. Data flow in a MapReduce processing

Traditional parallel applications are based on a runtime library for message passing such as MPI [5] and PVM [6] that have some programming features of communication and synchronization. The feature provided by a runtime library is a low-level primitive. In MapReduce, a programmer is able to focus on the problem that needs to be solved since only the map and reduce functions need to be implemented, and the framework takes care of computing while the programmer has to deal with lower-level mechanisms to control the data flow [1], [3].

2.2 Twister

There are some existing implementations of MapReduce like Hadoop[6] Twister is one of MapReduce implementations, which is an enhanced MapReduce runtime with an extended programming model that supports an iterative MapReduce computing efficiently [7]. In addition, it provides programming extensions to MapReduce with broadcast and scatter type for transferring data. These improvements allow Twister to support iterative MapReduce computations highly efficiently compared to other MapReduce runtimes. The demanding requirements have led to the development of a new programming model like Twister based MapReduce. It reads data from local disks of the worker nodes and handles the intermediate data in distributed memory of the worker nodes.

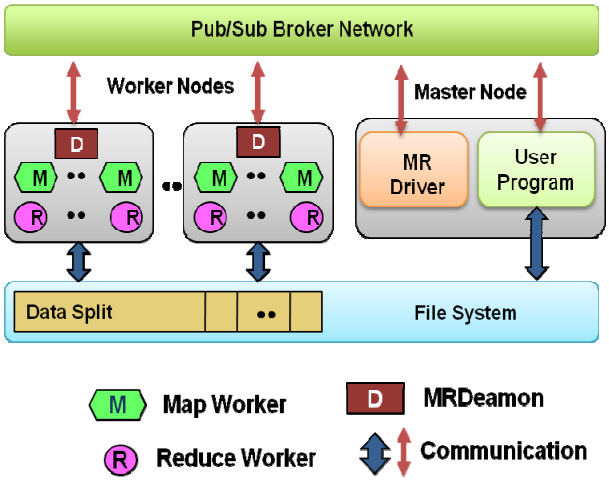

Fig. 2. Overall architecture of Twister

As shown in Fig. 2, all communication and data transfers are performed via a pub/sub broker network via NaradaBrokering that is an open-source, distributed messaging infrastructure [9].

The pub/sub paradigm applied to NaradaBrokering is a loose-coupled communication scheme for modeling the interaction between applications in distributed systems. Unlike the classic request/reply model, publish/subscribe provides time decoupling between message producers and consumers. A topic represents the event type that is used in a natural way to categorize the event. For example, once the event has occurred, the publisher as a producer can publish the event subscribing to a topic T can be viewed as becoming a member of a group T. Notification framework allows clients to register their interest in being notified of particular messages and supports an asynchronous, one way delivery of such notifications. Twister uses a pub/sub messaging infrastructure to handle four types of communication needs; (i) sending/receiving control events, (ii) sending data from the client side driver to the Twister daemons, (iii) transferring intermediate data between map and reduce tasks, and (iv) sending the outputs of the reduce tasks back to the client side driver to invoke the combine operation.

2.3 Docker

Virtualization provides a way to abstract the hardware and system resources from an operation system, which is used to reduce the actual number of physical servers and improve scalability and workloads in a cloud environment. A VM is a computing platform that creates a virtualized layer between the computing hardware and the application.

Docker is an open platform released by dotCloud as a container project for developers and system administrators to build, ship and run distributed applications. It automates the deployment of applications inside software containers by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux and enables applications to be quickly assembled from components and eliminated the friction. Especially virtualization technology is the key technology, though its deployment of cloud computing system can be implemented.

Docker is written in Go and the architecture of Docker is composed of a client and a server. Docker is composed of the daemon that sits on the server machine and accepts commands from the Docker client. The daemon communicates with the underlying OS through a libcontainer, which is a library that is able to run commands to manage the containers. The user communicates with the daemon through a Docker client, which can be a command line or a user interface. Finally, the registry is a service provided by the Docker platform on the cloud to host the libraries and the applications through an image. This image can be installed as a container through the Docker daemon and these packaged images can be created by any user and shared through the registry.

The Docker client is the user interface to Docker and user interact with the Docker daemon through the Docker client. Docker client can run on the same host with the Docker daemon or connect via sockets or a RESTful API. A Docker image is used to create Docker containers. It consists of a series of layers which Docker uses a union file system to combine these layers into a single and coherent image. The union file systems can build blocks for containers and the variants used for this task. As a result, Docker can be a lightweight container by adding or updating the specific layers rather than replacing the whole image. A Docker container is created from an image and includes an operating system, user-added files, and meta-data. In addition, it is an isolated and secure application platform that can run an application.

3. EXPERIMENTAL RESULTS

3.1 K-means clustering

K-means clustering is a method of cluster analysis, which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean. Given a data set, D, of n objects, and k, the number of clusters to form, a partitioning algorithm organizes the objects into k partitions (k <= n), where each partition represents a cluster. The error function used is the sum of the distance that each point is from its cluster's centroid. In the k-means MapReduce application we built, it is used to determine the number of iterations in the data set.

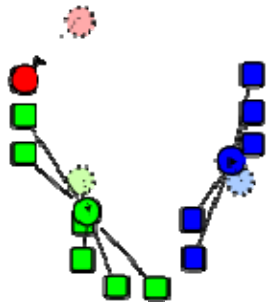

Fig. 3. Example of clustering by k-means

Fig, 3 shows how the centroid of each of the k clusters becomes the new mean. Given a set of observations ($X_1, X_2, \ldots, X_n$), where each observation is a d-dimensional real vector, k-means clustering aims to partition the n observations into k ($\leq n$) sets S = {S_1, S_2, …, S_k} so as to minimize the within-cluster sum of distance functions of each point in the cluster to the K center. Each of iteration generates a set of centroids that is updated. Equation 1 as an objective function is used to minimize the sum of the distance that each point is from its cluster's centroid:

$$\delta = \operatorname*{argmin}_{S} \sum_{i=1}^{k} \sum_{x \in S_i} \left\| x - \mu_i \right\|^2 \quad (1)$$

Where μ_i is the mean of points in S_i.

In the k-means MapReduce application we built, it is used to determine the number of iterations in the data set. We describe an overall framework of a MapReduce based k-means

with an initial starting configuration of this k-means clustering as input parameters. Fig. 4 shows the designed framework.

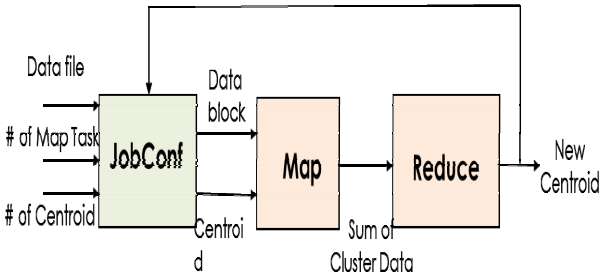

Fig. 4. Designed framework of a MapReduce based k-means

3.2 Experiment Environment

To evaluate in the designed framework, we conduct the Docker experiments running the k-means MapReduce application by using the following systems:

- Twister 0.8
- NaradaBrokering 4.2.2
- Linux 2.6.x

To build a testbed for Twister based MapReduce applications, we -took the following steps:

(1) Generate Twister image from a Dockerfile
(2) Instantiate a VM by the Twister image
(3) Configure a Twister middleware as well as pre-requite tools

In the first step of building Twister MapReduce, download and install tools including a JDK, openSSLserver and ant as well as Ubuntu operating system with Dockerfile. The following script shows about how to build the image.

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install software-properties-common -y
RUN add-apt-repository ppa:webupd8team/java -y
RUN apt-get update
RUN echo debconf shared/accepted-oracle-license-v1-1 select true | debconf-set-selections
RUN apt-get install oracle-java7-installer -y
RUN apt-get install openssh-server -y
RUN apt-get install ant -y
RUN apt-get -y install openssh-client
RUN ssh-keygen -q -t rsa -N '' -f /root/.ssh/id_rsa
RUN cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
ADD twister-0.8.tar /root/twister-0.8.tar
ADD NaradaBrokering-4.2.2.tar /root/NB.tar
EXPOSE 22
ENTRYPOINT service ssh start
```

Fig. 5 shows the image named ubuntu:twister_0.8 with c9023acbabc7 as identifier and 1.053 GB as size of image. Then we build an image named ubuntu:twister_0.8.

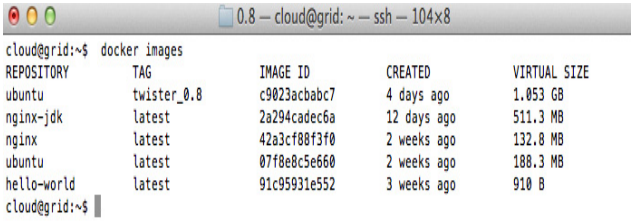

Fig. 5. Docker images

Fig. 6 shows the two Docker containers generated from the image ubuntu:twister. Each of these containers has a network interface with IP addresses 172.17.0.9 and 172.17.0.10 respectively.

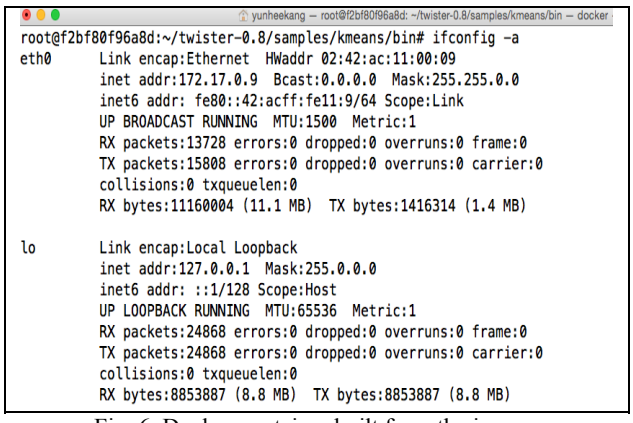

Fig. 6. Docker container built from the image

Fig. 7 shows the BrokerNode of NaradaBrokering that is used for transmitting intermediate data of a Twister application and TwisterDaemon that plays a role as a daemon for running a Twister application. In order to operate the Twister application, TwisterDaemon is running on two Docker containers. On the other hand, BrokerNode is running on one Docker container of them.

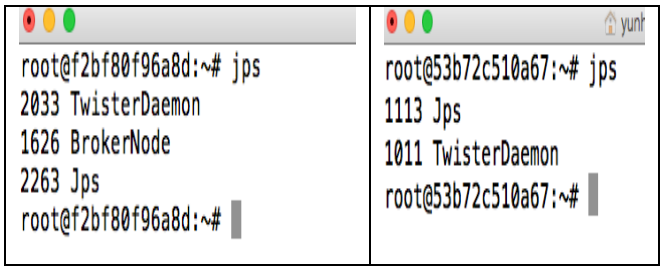

Fig. 7. Docker container that is running TwisterDaemon

The following shows the result of k-means MapReduce application. To evaluate performance of a k-means MapReduce application, we define a configuration of this experiment. This configuration is used to cluster 3,000 data points with 3 mapper tasks and 1 reducer task. The elapsed time is 15.646 second with 87 iterations. In the perspective of elapsed time, there is no difference between physical machine and virtual machine.

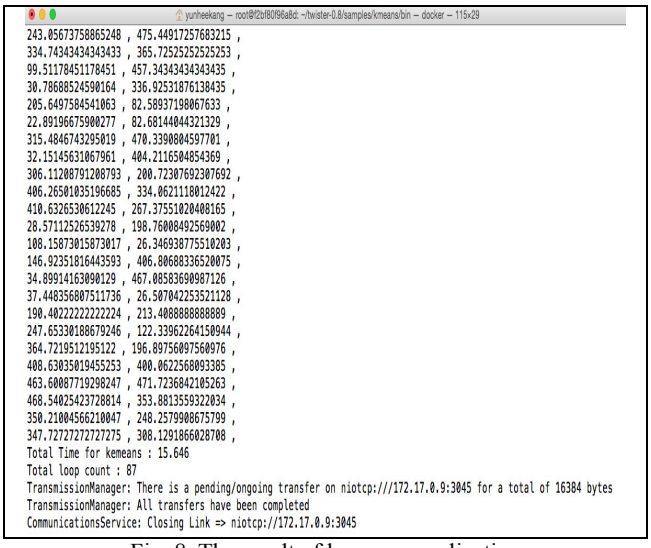

Fig. 8. The result of k-means application

4. CONCLUSION

Instead of relying on classical computing resources, we directed our attention to the use of virtual machines, where the characteristics of each physical node may be abstracted and controlled, thus simplifying the configuration process. This paper described the overall process of building the experimental environment for Twister applications. For this work we have chosen this specific tool due to the increasing popularity of MapReduce and cloud container technologies such as Docker. Basically this is the first attempt towards automatic configuration of Twister jobs on container-based cloud platform VM for many workloads. This paper aims at automatically configuring Twister workloads for container-driven clouds. In terms of elapsed time, there is no difference between physical machine and virtual machine.

ACKNOWLEDGEMENT

This work was supported by the research fund of Baekseok University (2016).

REFERENCES

[1] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud computing and grid computing 360-degree compared,” Proc. Of Grid computing environments workshop, 2008, p. 1.
[2] J. Dean and S. Ghemawat, “MapReduce: A Flexible Data Processing Tool,” Communications of the ACM, vol. 53, no. 1, Jan. 2010, pp. 72-77.
[3] K. Morton, A. Friesen, M. Balazinska, and D. Grossman, “Estimating the Progress of MapReduce Pipelines,” Proc. Of IEEE 26th International Conference on Data Engineering (ICDE), 2010, p. 681.
[4] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, Jan. 2008, pp. 107-113.
[5] MPI, MPI(Message Passing Interface). Available: http://www-unix.mcs.anl.gov/mpi/
[6] PVM, PVM(Parallel Virtual Machine). Available: http://www.csm.ornl.gov/pvm/
[7] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.H. Bae, J. Qiu, and G. Fox, “Twister: A Runtime for Iterative MapReduce,” Proc. of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), 2010, p. 810.
[8] G. Fox and S. Pallickara, “Deploying the NaradaBrokering Substrate in Aiding Efficient Web & Grid Service Interactions,” Proc. of the IEEE on Grid Computing, vol. 93, no. 3, 2005, p. 564.

Yunee Kang
He received the B.E. and M.S. degrees from Dongguk University, Seoul, Korea, in 1991, and 1993, respectively, and Ph.D. degree from Korea University, Seoul, Korea, in 2002, all in computer engineering. He is currently an Associative Professor with the Division of information and commutation, Baekseok University, Cheonan, Korea. His primary research interests lie broadly in distributed systems including fault-tolerance system, cloud computing and grid computing.