

Getting Started to Apache Hadoop

Running Hadoop Common, HDFS, Hadoop YARN, and Hadoop MapReduce on Multi Nodes

Ari Pratama Zhorifiandi

Computer Science Department

School of Electrical Engineering and Informatics

Institut Teknologi Bandung

Bandung, Indonesia

13514039@std.stei.itb.ac.id

Abstract— Hadoop has become everyone's big data darling. But it can only do so much, and savvy businesses need to make sure it's a good fit for their needs. In the past few years, Hadoop has earned a lofty reputation as the go-to big data analytics engine. To many, it's synonymous with big data technology. But the open source distributed processing framework isn't the right answer to every big data problem, and companies looking to deploy it need to carefully evaluate when to use Hadoop -- and when to turn to something else. In order to fulfil its needs, we should understand how to use Hadoop. In this paper, I will show how to build and configure Hadoop from source code.

Keywords—hadoop; distributed computing; hdfs; mapreduce; big data

I. INTRODUCTION

Hadoop is an open source, Java-based programming framework which is famous for supporting the processing and storage of tremendous large data sets in a distributed computing environment. It is part of the Apache project sponsored by the Apache Software Foundation.

Hadoop makes it possible to run applications on systems with thousands of commodity hardware nodes, and to handle thousands of terabytes of data. Its distributed file system facilitates rapid data transfer rates among nodes and allows the system to continue operating in case of a node failure. This approach lowers the risk of catastrophic system failure and unexpected data loss, even if a significant number of nodes become inoperative. Consequently, Hadoop quickly emerged as a foundation for big data processing tasks, such as scientific analytics, business and sales planning, and processing enormous volumes of sensor data, including from internet of things sensors.

II. BACKGROUND

As a software framework, Hadoop is composed of numerous functional modules. Basically, Hadoop uses Hadoop Common as a kernel to provide the framework's essential libraries. Other components include Hadoop Distributed File System (HDFS), which is capable of storing data across thousands of commodity

servers to achieve high bandwidth between nodes; Hadoop Yet Another Resource Negotiator (YARN), which provides resource management and scheduling for user applications; and Hadoop MapReduce, which provides the programming model used to tackle large distributed data processing -- mapping data and reducing it to a result.

HDFS has become a vital stuff for managing pools of big data and supporting big data analytics applications. Server failures are common because HDFS typically is deployed on low-cost commodity hardware. The file system is designed to be highly fault-tolerant, however, by facilitating the rapid transfer of data between compute nodes and enabling Hadoop systems to continue running if a node fails. That decreases the risk of catastrophic failure, even in the event that numerous nodes fail. It uses a master/slave architecture, with each cluster consisting of a single *NameNode* that manages file system operations and supporting *DataNodes* that manage data storage on individual compute nodes.

YARN is one of the key features in the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. Originally described by Apache as a redesigned resource manager, YARN is now characterized as a large-scale, distributed operating system for big data applications.

YARN is a software rewrite that decouples MapReduce's resource management and scheduling capabilities from the data processing component, enabling Hadoop to support more varied processing approaches and a broader array of applications. YARN combines a central resource manager that reconciles the way applications use Hadoop system resources with node manager agents that monitor the processing operations of individual cluster nodes.

Hadoop enables resilient, distributed processing of massive unstructured data sets across commodity computer clusters, in which each node of the cluster includes its own storage. MapReduce serves two essential functions: It parcels out work to various nodes within the cluster or map, and it organizes and

reduces the results from each node into a cohesive answer to a query. To distribute input data and collate results, MapReduce operates in parallel across massive cluster sizes. Because cluster size doesn't affect a processing job's final results, jobs can be split across almost any number of servers.

III. BUILDING FROM SOURCE CODE

Building Hadoop from source code is essential for further development. We can modify how Hadoop works after we successfully build it from source code. First step you need to do is clone the git repository of Apache Hadoop. You can find it here <https://github.com/apache/hadoop>.

After completely clone the repository, You need to meet these requirements:

- * Unix System
- * JDK 1.6
- * Maven 3.1.1
- * ProtocolBuffer 2.4.1+ (for MapReduce and HDFS)
- * CMake 2.6 or newer (if compiling native code)
- * Internet connection for first build (to fetch all Maven and Hadoop dependencies)

Please make sure you're using the right version, because different version could cause to unusual errors while building. Then you can run one of these command on your Hadoop directory

- * Clean : mvn clean [-Preleasedocs]
- * Compile : mvn compile [-Pnative]
- * Run tests : mvn test [-Pnative] [-Pshelltest]
- * Create JAR : mvn package
- * Run findbugs : mvn compile findbugs:findbugs
- * Run checkstyle : mvn compile checkstyle:checkstyle
- * Install JAR in M2 cache : mvn install
- * Deploy JAR to Maven repo : mvn deploy
- * Build distribution : mvn package
- * Change Hadoop version : mvn versions:set

I create binary distribution without native code and without documentation, then I used this command:

```
$sudo mvn package -Pdist,native -DskipTests -Dtar
```

It will take several time to finish building it. Different OS configuration could lead some trouble while building Hadoop. If you find some problem, you can find the solution in here <https://issues.apache.org/jira/browse/HADOOP/>.

IV. SETTING UP MULTI NODE CLUSTER

After finishing build hadoop, we can move forward to setup multi node cluster for Hadoop. You can find installed Hadoop in [Your Clone Directory]/hadoop-dist/target/hadoop-2.7.2. Then, try the following command:

```
$bin hadoop
```

This will display the usage documentation for the hadoop script. Basically, Hadoop are already supported to Single Nodes Cluster and ready to startHadoop cluster in one of the three supported modes:

- Local (Standalone) Mode
- Pseudo-Distributed Mode
- Fully-Distributed Mode

But, We will go through multi node cluster. First, we need to set up a RSA public/private key pair to be able to ssh into the node. On each node (both master and slave), type the following commands to generate RSA key pair of the node.

```
$ssh-keygen -t rsa -P "" -f $HOME/.ssh/id_rsa
```

The private key is stored in the file specified by the -f option, in this case is \$HOME/.ssh/id_rsa, and the public key is stored in the file with the same name but with a .pub extension appended, in this case will be \$HOME/.ssh/id_rsa.pub.

Then, we need to make sure that the public key is authorized by copying the public key into \$HOME/.ssh/authorized_keys using the following command.

```
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

We need to make sure that host names in the cluster are being configured correctly in the host file at /etc/host so that each node can be communicated by its name rather than its IP-address. We then ssh to localhost machine and the actual host names to make sure that ssh is working correctly. Both the master node and the slave node must be able to ssh to each other. This step will also add the hosts' fingerprint into the known host file.

```
$ ssh localhost  
$ ssh zhorifiandi@Zhorifiandi.local
```

Finally, we need to distribute the public key of the master node to all slave nodes in the cluster by using the following command to append the public key to a remote host.

```
$ cat $HOME/.ssh/id_rsa.pub | ssh USERNAME@HOST_NAME 'cat >> $HOME/.ssh/authorized_keys'
```

After that, you need to edit following config files in your Hadoop directory

1) etc/hadoop/core-site.xml

```
<configuration>  
<property>  
<name>fs.default.name</name>  
<value>hdfs://Zhorifiandi.local:9000</value>  
</property><property>  
<name>hadoop.tmp.dir</name>  
<value>/tmp/hadoop-`${user.name}`</value>  
</property>  
</configuration>
```

2) etc/hadoop/hdfs-site.xml

```
<configuration>
<property>
<name>dfs.replication</name>
<value>4</value>
</property><property>
<name>dfs.permissions</name>
<value>>false</value>
</property>
</configuration>
```

3) etc/hadoop/mapred-site.xml

```
<configuration>
<property>
<name>mapred.job.tracker</name>
<value>zhorifiandi@Zhorifiandi.local:9001</value>
>
</property><property>
<name>mapred.tasktracker.map.tasks.maximum</name>
>
<value>1</value>
</property><property>
<name>mapred.tasktracker.reduce.tasks.maximum</n
ame>
<value>1</value>
</property><property>
<name>mapred.max.split.size</name>
<value>1000</value>
</property>
</configuration>
```

4) etc/hadoop/yarn-site.xml

```
<configuration>
<property>
<name>yarn.nodemanager.aux-services
</name>
<value>mapreduce_shuffle</value>
</property>
</configuration>
```

5) etc/hadoop/masters

zhorifiandi@Zhorifiandi.local

6) etc/hadoop/slaves

localhost

V. EXECUTION

Before starting our Hadoop cluster, we need to initialize the HDFS first by using the following command.

```
$ bin/Hadoop namenode -format
```

Then, It's time to start our Hadoop cluster, just execute the following command:

```
$ sbin/start-all.sh
```

We can test whether hadoop already working or not by running a job. I run mapreduce grep job by these following commands:

```
$ bin/hdfs dfs -mkdir /user
$ bin/hdfs dfs -mkdir /user/zhorifiandi
$ bin/hdfs dfs -put etc/hadoop input
$ bin/hadoop jar share/hadoop/mapreduce/hadoop-
mapreduce-examples-2.7.2.jar grep input output 'dfs[a-
z.]+'
$ bin/hdfs dfs -cat output
```

Then, you'll get this result:

```
Zhorifiandi:hadoop-2.7.2 zhorifiandi$ bin/hdfs dfs -cat output/*
6 dfs.audit.logger
4 dfs.class
3 dfs.server.namenode.
2 dfs.period
2 dfs.audit.log.maxfilesize
2 dfs.audit.log.maxbackupindex
1 dfsmetrics.log
1 dfsadmin
1 dfs.servers
1 dfs.replication $ bin/hdfs dfs -get output output
1 dfs.permissions $ cat output/*
1 dfs.file
```

Fig. 1. Result of Running hadoop job

After finish running the job, you'll need to stop all nodes by running this command:

```
$ sbin/stop-all.sh
```

CONCLUSION

This paper has shown you how to configure and running Hadoop on multiple nodes, but it still operate in localhost. In further exploration, I will show you how to run it on more nodes. It can be run on a network-testbed like Emulab.

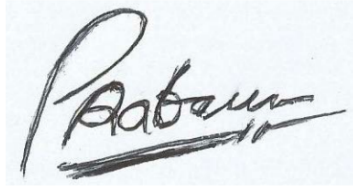
REFERENCES

- [1] Tom White, Hadoop: The Definitive Guide, O'Reilly Media, Inc., 2009
- [2] <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html>
- [3] http://boatboat001.com/index.php/blogs/view/setting_up_a_hadoop_cluster_under_mac_os_x_mountain

STATEMENT

I hereby declare that paper that I wrote is my own writing, not adaptation, or translation from someone else's paper, and not plagiarism.

Bandung, May 3rd 2017

A handwritten signature in black ink on a light blue background. The signature is written in a cursive style and appears to read 'Pratama'.

Ari Pratama Zhorifiandi

13514039