Cloud computing with Linux and Apache Hadoop

Yi Ming Huang
Zhao Hui Nie

Many companies like IBM®, Google, VMWare, and Amazon have provided products and strategies for Cloud computing. This article shows you how to use Apache Hadoop to build a MapReduce framework to make a Hadoop Cluster and how to create a sample MapReduce application which runs on Hadoop. You will also learn how to set up a time/disk-consuming task on the cloud.

Introduction to cloud computing

Recently there has been increasing hype about cloud computing, which is regarded as the next trend of the IT industry. Cloud computing can be loosely defined as using scalable computing resources provided as a service from outside your environment on a pay-per-use basis. You can access any of the resources that live in the "cloud" across the Internet and don't have to worry about computing capacity, bandwidth, storage, security, and reliability.

This article briefly introduces cloud computing platforms like Amazon EC2, on which you can rent virtual Linux® servers, and then introduces an open source MapReduce framework named Apache Hadoop, which will be built onto the virtual Linux servers to establish the cloud computing framework. However, Hadoop is not restricted to be deployed on VMs hosted by any vendor; you can also deploy it on normal Linux OS on physical machines.

Before we dive into Apache Hadoop, we will give a brief introduction to the structure of the cloud computing system. Figure 1 is a view of the layers of cloud computing and some existing offerings. You can reference the Resources section for more details about the layers of cloud computing.

The infrastructure (Infrastructure-as-a-Service, or IaaS) is the leasing of infrastructure (computing resources and storage) as a service. IaaS provides the capability for a user to lease a computer (or virtualized host) or data center with specific quality-of-service constraints that has the ability to execute certain operating systems and software. Amazon EC2 is playing a role as the IaaS in these layers and provides users virtualized hosts. The Platform (Platform-as-a-Service, or PaaS) focuses on the software framework or services, which provide the ability of APIs to "cloud" computing on the infrastructure. Apache Hadoop plays a role as PaaS and will be built on the virtualized hosts as the cloud computing platform.
Figure 1. Layers of cloud computing and existing offerings

Amazon EC2

Amazon EC2 is a Web service that lets you request virtual machines with various capacities (CPU, disks, memory, and more). You pay for only the computing time you use while leaving the hosting chores to Amazon.

These instances, Amazon Machine Images (AMIs), are based on Linux and can run any application or software you want. After you have rented the servers from Amazon, you can use normal SSH tools to set up connection and manipulate your servers just like physical ones.

A more detailed introduction of EC2 is out of the scope of this article. See the Resources section for additional information.

The best practice to deploy a Hadoop cloud computing framework is to deploy it on the AMIs, which can utilize the cloud capacity when computing capability, bandwidth, storage, and more are not issues. However, in the next part of this article, we will build Hadoop onto VMWare images of Linux servers hosted locally, since Hadoop is not restricted to be deployed on any cloud solution. Before that, we will give some introduction on Apache Hadoop.

Apache Hadoop

Apache Hadoop is a software framework (platform) that enables a distributed manipulation of vast amount of data. Introduced in 2006, it is supported by Google, Yahoo!, and IBM, to name a few. You can think it as a model of PaaS.

At the heart of its design is the MapReduce implementation and HDFS (Hadoop Distributed File System), which was inspired by the MapReduce (introduced by a Google paper) and the Google File System.

MapReduce

MapReduce is a software framework introduced by Google that supports distributed computing on large data sets on clusters of computers (or nodes). It is the combination of two processes named Map and Reduce.

In the Map process, the master node takes the input, divides it up into smaller sub-tasks, and distributes those to worker nodes.
The worker node processes that smaller task, and passes the answer back to the master node.

In the Reduce process, the master node then takes the answers of all the sub-tasks and combines them to get the output, which is the result of the original task.

Refer to Figure 2, which provides a conceptual idea about the MapReduce flow.

The advantage of MapReduce is that it allows for the distributed processing of the map and reduction operations. Because each mapping operation is independent, all maps can be performed in parallel, thus reducing the total computing time.

**HDFS**

The complete introduction to HDFS and how to operate on it is beyond the scope of this article. See the Resources section for additional information.

From the perspective of an end user, HDFS appears as a traditional file system. You can perform CRUD actions on files with certain directory path. But, due to the characteristics of distributed storage, there are "NameNode" and "DataNode," which take each of their responsibility.

The NameNode is the master of the DataNodes. It provides metadata services within HDFS. The metadata indicates the file mapping of the DataNode. It also accepts operation commands and determines which DataNode should perform the action and replication.

The DataNode serves as storage blocks for HDFS. They also respond to commands that create, delete, and replicate blocks received from the NameNode.

**JobTracker and TaskTracker**

When an application is submitted, input and output directories contained in the HDFS should be provided. The JobTracker, as the single control point for launching the MapReduce applications, decides how many TaskTracker and subordinate tasks to be created and then assigns each sub-task to TaskTracker. Each TaskTracker reports status and completed tasks back to the JobTracker.

Usually one master node acts as the NameNode and JobTracker and the slave acts as the DataNode and TaskTracker. The conceptual view of Hadoop cluster and the follow of MapReduce are shown in Figure 2.
Set up Apache Hadoop

Now we will set up the Hadoop Cluster on the Linux VMs and then we can run MapReduce applications on the Hadoop Cluster.

Apache Hadoop supports three deployment modes:

- **Standalone Mode**: By default, Hadoop is configured to run in a non-distributed standalone mode. This mode is useful to debug your application.
- **Pseudo-distributed Mode**: Hadoop can also be run in a single node pseudo-distributed mode. In this case, each Hadoop daemon is running as a separate Java™ process.
- **Fully-distributed Mode**: Hadoop is configured on different hosts and run as a cluster.

To set up Hadoop in standalone or pseudo-distributed mode, refer to the Hadoop Web site for reference. In this article, we will only cover setting up Hadoop in fully-distributed mode.

**Prepare the environment**

In this article, we need three GNU/Linux servers; one will work as a master node and the other two will be slave nodes.

**Table 1. Server information**

<table>
<thead>
<tr>
<th>Server IP</th>
<th>Server Host Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.30.210.159</td>
<td>Vm-9-30-210-159</td>
<td>Master (NameNode and JobTracker)</td>
</tr>
</tbody>
</table>

Each machine needs to have Java SE 6 installed as well as the Hadoop binary. See the Resources section for more information. This article uses Hadoop version 0.19.1.

You also need SSH installed and ssdh running on each machine. Popular Linux releases like SUSE and RedHat have them installed by default.
Set up communications

Update the /etc/hosts file and make sure the three machines can reach each other using IP and hostname.

Because the Hadoop master node communicates with slave nodes using SSH, you should set up an authenticated no-passphrase SSH connection between the master and slaves. On each machine, execute the following command to generate the RSA public and private keys.

```bash
ssh-keygen -t rsa
```

This will generate id_rsa.pub under the /root/.ssh directory. Rename the master's id_rsa.pub (59_rsa.pub in this case) and copy it to slave nodes. Then execute the following command to add the master's public key to the slaves' authorized keys.

```bash
cat /root/.ssh/59_rsa.pub >> /root/.ssh/authorized_keys
```

Now try to SSH the slave nodes. It should be connected without needing a password.

Set up the master node

Set up Hadoop to work in a fully-distributed mode by configuring the configuration files under the <Hadoop_home>/conf/ directory.

Configure the Hadoop deployment in hadoop-site.xml. This configuration overrides the configurations in hadoop-default.xml.

### Table 2. Configuration property

<table>
<thead>
<tr>
<th>Property</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fs.default.name</td>
<td>NameNode URI</td>
</tr>
<tr>
<td>mapred.job.tracker</td>
<td>JobTracker URI</td>
</tr>
<tr>
<td>dfs.replication</td>
<td>Number of replication</td>
</tr>
<tr>
<td>hadoop.tmp.dir</td>
<td>Temp directory</td>
</tr>
</tbody>
</table>

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.default.name</name>
  </property>
  <property>
    <name>mapred.job.tracker</name>
  </property>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```
Configure the hadoop-env.sh file to specify the JAVA_HOME. Comment out the line and specify your JAVA_HOME directory.

```bash
export JAVA_HOME=<JAVA_HOME_DIR>
```

Add the master node IP address to the master file.

9.30.210.159

Add the slave node IP addresses to the slave file.


**Set up the slave nodes**
Copy the hadoop-site.xml, hadoop-env.sh, masters, and slaves to each slave nodes; you can use SCP or another copy utility.

**Format the HDFS**
Run the following command to format the Hadoop distributed file system to initialize.

```bash
<@Hadoop_home>/bin/hadoop namenode -format
```

**Verify the Hadoop Cluster**
Now you can start the Hadoop cluster using `bin/start-all.sh`. The command output indicates some logs located on the master and slaves. Verify the logs and make sure everything is correct. If you mess up something, you can format the HDFS and clear the temp directory specified in hadoop-site.xml and start again.

Visit the following URL to verify that the master and other slave nodes are healthy.


Now you have set up the Hadoop Cluster on the cloud, and it's ready to run the MapReduce applications.

**Create a MapReduce application**
MapReduce applications must have the characteristic of "Map" and "Reduce," meaning that the task or job can be divided into smaller pieces to be processed in parallel. Then the result of each sub-task can be reduced to make the answer for the original task. One example of this is Website keyword searching. The searching and grabbing tasks can be divided and delegated to slave
nodes, then each result can be aggregated and the outcome (the final result) is on the master node.

**Try the sample application**

Hadoop comes with some sample applications for testing. One of them is a word counter, which counts for certain word occurrence in several files. Run this application to verify the Hadoop Cluster.

First, put the input files (under the conf/ directory) in the distributed file system. We will count the words in these files.

```
$ bin/hadoop fs -put conf input
```

Then, run the sample, which counts occurrences of words that start with "dfs."

```
$ bin/hadoop jar hadoop-*-examples.jar grep input output 'dfs[a-z.]+'
```

The output of the command indicates the Map and Reduce process.

The previous two commands will generate two directories under HDFS, one "input" and one "output." You can list them with:

```
$ bin/hadoop fs -ls
```

View the files that have been output on the distributed file system. It lists the occurrence of words starting with "dfs*" by key-value pairs.

```
$ bin/hadoop fs -cat ouput/*
```

Now visit the JobTracker site to see a completed job log.

**Create a Log Analyzer MapReduce application**

Now create a Portal (IBM WebSphere® Portal v6.0) Log Analyzer application that has much in common with the WordCount application in Hadoop. The Analyzer will go through all the Portal's SystemOut*.log files, and show how many times the applications on Portal have been started during a certain time period.

In a Portal environment, all the logs will be split to 5MB pieces and they are good candidates to be analyzed by several nodes in parallel.

**hadoop.sample.PortalLogAnalyzer.java**

```java
public class PortalLogAnalyzer {

    public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {

        private static String APP_START_TOKEN = "Application started:'",
        private Text application = new Text();
```
```java
public void map(LongWritable key, Text value,
    OutputCollector<Text, IntWritable> output,
    Reporter reporter) throws IOException {

    String line = value.toString();
    if (line.indexOf(APP_START_TOKEN) > -1) {
        int startIndex = line.indexOf(APP_START_TOKEN);
        startIndex += APP_START_TOKEN.length();
        String appName = line.substring(startIndex).trim();
        application.set(appName);
        output.collect(application, new IntWritable(1));
    }
}

public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output,
        Reporter reporter) throws IOException {

        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}

public static void main(String[] args) throws IOException {
    JobConf jobConf = new JobConf(PortalLogAnalyzer.class);
    jobConf.setJobName("Portal Log Analizer");
    jobConf.setOutputKeyClass(Text.class);
    jobConf.setOutputValueClass(IntWritable.class);
    jobConf.setMapperClass(Map.class);
    jobConf.setCombinerClass(Reduce.class);
    jobConf.setReducerClass(Reduce.class);
    jobConf.setInputFormat(TextInputFormat.class);
    jobConf.setOutputFormat(TextOutputFormat.class);
    FileInputFormat.setInputPaths(jobConf, new Path(args[0]));
    FileOutputFormat.setOutputPath(jobConf, new Path(args[1]));
    JobClient.runJob(jobConf);
}
```

Refer to the Hadoop site's API document for a complete explanation of Hadoop API. Here is a brief description.

The **Map class** implements the map function, which goes through each line of the log file and gets the application's name. Then put the application name in the output collection as a key-value pair.

The **Reduce class** sums up all the values that have the same key (same application name). Thus, the output of this application will be key-value pairs that indicate how many times each application on Portal has been started.

The **Main function** configures the MapReduce job and runs it.
Run the PortalLogAnalyzer

First, copy the Java code to the Master node and compile it. Copy the Java code to `<hadoop_home>/workspace` directory. Compile it and archive it in a Jar file, which will be run with the `hadoop` command later.

```bash
$ mkdir classes
$ javac -cp ../hadoop-0.19.1-core.jar -d classes hadoop/sample/PortalLogAnalyzer.java
$ jar -cvf PortalLogAnalyzer.jar -C classes/ .
```

Copy your Portal logs to `workspace/input`. Let's suppose we have several log files that contain all the logs in May 2009. Put these logs to the HDFS.

```bash
$ bin/hadoop fs -put workspace/input input2
```

When you run the PortalLogAnalyzer, the output indicates the process of Map and Reduce.

```bash
$ bin/hadoop jar workspace/PortalLogAnalizer.jar hadoop.sample.PortalLogAnalizer input2 output2
```

**Figure 3. Output of the task**

```
INFO mapred.FileInputFormat: Total input paths to process : 1
INFO mapred.JobClient: Running job: job_200905262355_0005
INFO mapred.JobClient: map 0 % reduce 0 %
INFO mapred.JobClient: map 50 % reduce 0 %
INFO mapred.JobClient: map 100 % reduce 0 %
INFO mapred.JobClient: map 100 % reduce 100 %
INFO mapred.JobClient: Job complete: job_200905262355_0005
INFO mapred.JobClient: Counters: 17
INFO mapred.JobClient: Map Reduce Framework
INFO mapred.JobClient: Reduce input groups=125
INFO mapred.JobClient: Combine output records=112
INFO mapred.JobClient: Map output records=125
INFO mapred.JobClient: Reduce output records=125
INFO mapred.JobClient: Map output bytes=10572
INFO mapred.JobClient: Map output bytes=1113632
INFO mapred.JobClient: Combine input records=112
INFO mapred.JobClient: Reduce input records=512
```

After the application finishes, the output should be similar to Figure 4, below.

```bash
$ bin/hadoop fs -cat output2/*
```
Figure 4. Partial output

When you visit the JobTracker site, you will see another completed job. Notice the last line in Figure 5.

Figure 5. Completed jobs

<table>
<thead>
<tr>
<th>JobId</th>
<th>Priority</th>
<th>Name</th>
<th>Map % Complete</th>
<th>Map Total</th>
<th>Reduce % Complete</th>
<th>Reduce Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job_000000000000000001</td>
<td>normal</td>
<td>grep-search</td>
<td>100.0%</td>
<td>11</td>
<td>100.0%</td>
<td>1</td>
</tr>
<tr>
<td>Job_000000000000000002</td>
<td>normal</td>
<td>grep-sort</td>
<td>100.0%</td>
<td>1</td>
<td>100.0%</td>
<td>1</td>
</tr>
<tr>
<td>Job_000000000000000003</td>
<td>normal</td>
<td>PortalLosAnalyser</td>
<td>100.0%</td>
<td>2</td>
<td>100.0%</td>
<td>1</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2009
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)