**Practical 11: Hadoop Configuration – HDFS Administering –Monitoring & Maintenance.**

Hadoop is an open source framework for storing and distributed batch processing of huge datasets on clusters of commodity hardware. Hadoop can be used on a single machine (Standalone Mode) as well as on a cluster of machines (Distributed Mode – Pseudo & Fully). One of the striking features of Hadoop is that it efficiently distributes large amounts of work across a cluster of machines/commodity hardware.

Standalone Mode- In standalone mode, we will configure Hadoop on a single machine (e.g. an Ubuntu machine on the host VM). The configuration in standalone mode is quite straightforward and does not require major changes.

Pseudo-Distributed Mode- In a pseudo distributed environment, we will configure more than one machine, one of these to act as a master and the rest as slave machines/node. In addition we will have more than one Ubuntu machine playing on the host VM.

Fully Distributed Mode- It is quite similar to a pseudo distributed environment with the exception that instead of VM the machines/node will be on a real distributed environment.

Following are some of the prerequisites for configuring Hadoop:

Hadoop requires Java 1.5+ installation. However, using Java 1.6 is recommended for running Hadoop. It can be run on both Windows & Unix but Linux/Unix  best support the production environment. Working with Hadoop on Windows also requires Cygwin installation.

**Installing & Configuring Hadoop in Standalone Mode**

You might want to create a dedicated user for running Apache Hadoop but it is not a prerequisite. In our demonstration, we will be using a default user for running Hadoop.

**Environment**

Ubuntu 10.10

JDK 6 or above

Hadoop-1.1.2 (Any stable release)

Follow these steps for installing and configuring Hadoop on a single node:

**Step-1. Install Java**

In this tutorial, we will use Java 1.6 therefore describing the installation of Java 1.6 in detail.

Use the below command to begin the installation of Java

|  |  |
| --- | --- |
| 1 | $ sudo apt-get install openjdk-6-jdk |

or

|  |  |
| --- | --- |
| 1 | $ sudo apt-get install sun-java6-jdk |

This will install the full JDK under /usr/lib/jvm/java-6-sundirectory.

**Step-2. Verify Java installation**

You can verify java installation using the following command

|  |  |
| --- | --- |
| 1 | $ java –version |

On executing this command, you should see output similar to the following:

java version “1.6.0\_27”

Java(TM) SE Runtime Environment (build 1.6.0\_45-b06)

Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)

**Step-3. SSH configuration**

Install SSH using the command.

|  |  |
| --- | --- |
| 1 | sudo apt-get install ssh |

Generate ssh key  
ssh -keygen -t rsa -P “” (press enter when asked for a file name; this will generate a passwordless ssh file)

Now copy the public key (id\_rsa.pub) of current machine to authorized\_keysBelow command copies the generated public key in the .ssh/authorized\_keys file:

|  |  |
| --- | --- |
| 1 | cat $HOME/.ssh/id\_rsa.pub >> $HOME/.ssh/authorized\_keys |

Verify ssh configuration using the command

|  |  |
| --- | --- |
| 1 | ssh localhost |

Pressing yes will add localhost to known hosts

**Step-4. Download Hadoop**

Download the latest stable release of Apache Hadoop from <http://hadoop.apache.org/releases.html>.

Unpack the release tar – zxvf hadoop-1.0.3.tar.gz

Save the extracted folder to an appropriate location, HADOOP\_HOME will be pointing to this directory.

**Step-5. Verify Hadoop**

Check if the following directories exist under HADOOP\_HOME: bin, conf, lib, bin

Use the following command to create an environment variable that points to the Hadoop installation directory (HADOOP\_HOME)

|  |  |
| --- | --- |
| 1 | export HADOOP\_HOME=/home/user/hadoop |

Now place the Hadoop binary directory on your command-line path by executing the command

|  |  |
| --- | --- |
| 1 | export PATH=$PATH:$HADOOP\_HOME/bin |

Use this command to verify your Hadoop installation:

hadoop version

The o/p should be similar to below one

Hadoop 1.1.2

Subversion <https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20> -r911707

Compiled by chrisdo on Fri Feb 19 08:07:34 UTC 2010

**Step-6. Configure JAVA\_HOME**

Hadoop requires Java installation path to work on, for this we will be setting JAVA\_HOME environment variable and this will point to our Java installation dir.

Java\_Home can be configured in ~/.bash\_profile or ~/.bashrc file. Alternatively you can also let hadoop know this by setting  Java\_Home  in hadoop **conf/hadoop-env.sh** file.

Use the below command to set JAVA\_HOME on Ubuntu

|  |  |
| --- | --- |
| 1 | export JAVA\_HOME=/usr/lib/jvm/java-6-sun |

JAVA\_HOME can be verified by command

|  |  |
| --- | --- |
| 1 | echo $JAVA\_HOME |

**Step-7. Create Data Directory for Hadoop**

An advantage of using Hadoop is that with just a limited number of directories you can set it up to work correctly. Let us create a directory with the name hdfs and three sub-directories name, data and tmp.

Since a Hadoop user would require to read-write to these directories you would need to change the permissions of above directories to 755 or 777 for Hadoop user.

**Step-8. Configure Hadoop XML files**

Next, we will configure Hadoop XML file.  Hadoop configuration files are in the  HADOOP\_HOME/conf dir.

**conf/core-site.xml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <!--?xml version="1.0"-->>  <!--?xml -stylesheet type="text/xsl" href="configuration.xsl"?-->  <! -- Putting site-specific property overrides the file. -->    fs.default.name  hdfs://localhost:9000    hadoop.temp.dir  /home/girish/hdfs/temp<span style="font-family: Georgia, 'Times New Roman', 'Bitstream Charter', Times, serif; font-size: 13px; line-height: 19px;"> </span> |

**conf/hdfs-site.xml**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <! -- Putting site specific property overrides in the file. -->    dfs.name.dir  /home/girish/hdfs/name    dfs.data.dir  /home/girish/hdfs/data    dfs.replication  1 |  |
| 1 | <strong style="font-family: Georgia, 'Times New Roman', 'Bitstream Charter', Times, serif; font-size: 13px; line-height: 19px;">conf/mapred-site.xml</strong> | |

|  |  |
| --- | --- |
| 1  2  3  4 | <! -- Putting site-specific property overrides this file. -->    mapred.job.tracker  localhost:9001 |

**conf/masters**

Not required in single node cluster.  
**conf/slaves**  
Not required in single node cluster.

**Step-9. Format Hadoop Name Node-**

Execute the below command from hadoop home directory

|  |  |
| --- | --- |
| 1 | $ ~/hadoop/bin/hadoop namenode –format |

The following image gives an overview of a Hadoop Distributed File System Architecture.

**Step-10. Start Hadoop daemons**

|  |  |
| --- | --- |
| 1 | $ ~/hadoop/bin/start-all.sh |

**Step-11. Verify the daemons are running**

|  |  |
| --- | --- |
| 1 | $ jps  (if jps is not in path, try  /usr/java/latest/bin/jps) |

output will look similar to this

9316 SecondaryNameNode

9203 DataNode

9521 TaskTracker

9403 JobTracker

9089 NameNode

Now we have all the daemons running:

Note: If your master server fails to start due to the dfs safe mode issue, execute this on the Hadoop command line:

|  |  |
| --- | --- |
| 1 | hadoop dfsadmin -safemode leave |

Also make sure to format the namenode again if you make changes to your configuration.

**Step-12. Verify UIs by namenode & job tracker**

Open a browser window and type the following URLs:

**namenode UI**:   http://machine\_host\_name:50070

**job tracker UI**:   http://machine\_host\_name:50030

substitute ‘machine host name’ with the public IP of your node e.g:  http://localhost:50070

Now you have successfully installed and configured Hadoop on a single node.

**BASIC HADOOP ADMIN COMMANDS**

(Source: Getting Started with Hadoop):

The ~/hadoop/bin directory contains some scripts used to launch Hadoop DFS and Hadoop Map/Reduce daemons. These are:

start-all.sh – Starts all Hadoop daemons, the namenode, datanodes, the jobtracker and tasktrackers.

stop-all.sh – Stops all Hadoop daemons.

start-mapred.sh – Starts the Hadoop Map/Reduce daemons, the jobtracker and tasktrackers.

stop-mapred.sh – Stops the Hadoop Map/Reduce daemons.

start-dfs.sh – Starts the Hadoop DFS daemons, the namenode and datanodes.

stop-dfs.sh – Stops the Hadoop DFS daemons.

**EXECUTING WORDCOUNT EXAMPLE IN HADOOP STANDALONE MODE**

When you download Hadoop, it comes with some existing demonstration programs and WordCount is one of them.

**Step-1. Creating a working directory for your data**

create a directory and name it *dft*.

|  |  |
| --- | --- |
| 1  2  3  4  5 | $ mkdir dft    $ cd dft    ~/dft$ |

**Step-2. Creating a working directory for data**

To process our text file we will have to provide this file to Hadoop File System (HDFS) afterwards Hadoop namenode and datanode willl be sharing this file from HDFS.

**1.    Creating a local copy**

Create your own text file with some commonly used words.

Let us give it a file name of MyTextFile.txt

**2.    Copy Data File to HDFS**

Copy the data file MyTextFile.txt to the **Hadoop File System** (HDFS):

**syntax:**hadoop dfs -copyFromLocal

|  |  |
| --- | --- |
| 1 | $ .bin/hadoop dfs -copyFromLocal /home/hadoop/dft dft |

Follow the below steps, if you encounter any issues e.g. “Cannot access dft: No such file or directory.”

Check the Hadoop dfs directory to see if the file already exists.

[user@ubuntu](mailto:user@ubuntu):/opt/hadoop-1.2.1$ **./bin/hadoop dfs -ls dft**

Found 1 items

-rw-r–r–   1 username supergroup    1573078 2013-10-09 00:32

/user/username/dft/MyTextFile.txt

If the file already exists it needs to be deleted first

[user@ubuntu](mailto:user@ubuntu):/opt/hadoop-1.2.1$ **./bin/hadoop dfs -rmr dft**

Deleted hdfs://localhost:9000/user/username/dft

Re run the -copyFromLocal command

[user@ubuntu](mailto:user@ubuntu):/opt/hadoop-1.2.1$ **.bin/hadoop dfs -copyFromLocal /home/hadoop/dft dft**

**3.    Confirm Data File is available at HDFS**

|  |  |
| --- | --- |
| 1 | $ hadoop dfs –ls |

Found x items … drwxr-xr-x – hadoop supergroup 0 2010-03-16 11:36 /user/hadoop/dft

Verify that your directory is now in the Hadoop File System, as indicated above.

**4.    Check the contents of your directory:**

|  |  |
| --- | --- |
| 1 | $ hadoop dfs -ls dft |

Found 1 items -rw-r–r– 2 hadoop supergroup 1573044 2010-03-16 11:36

/user/hadoop/dft/MyTextFile.txt

Verify that the file MyTextFile.txt exists.

**Step-3. WordCount.java Map-Reduce Program**

The program has several sections:

**THE MAP SECTION**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | public static class MapClass extends MapReduceBase implements Mapper<longWritable, Text, Text, IntWritable> {      private final static IntWritable one = new IntWritable(1);      private Text word = new Text();      public void map(LongWritable key, Text value, OutputCollector<text, IntWritable> output, Reporter reporter) throws IOException {          String line = value.toString();          StringTokenizer tokenizer = new StringTokenizer(line);          while (tokenizer.hasMoreTokens()) {              word.set(tokenizer.nextToken());              output.collect(word, one);          }}  } |

Hadoop breaks the text file in parts which now becomes the input for mapper. Then Hadoop will tokenize each line and tags each word as a datagram <”word”, 1> which indicates this word has appeared once, so if a particular word appeared 10 time, the datagram <”word”, 1> will be appear 10 times as well to reflect the repetition.

**THE REDUCE SECTION**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | public static class Reduce extends MapReduceBase implements Reducer<text, IntWritable, Text, IntWritable> {   public void reduce(Text key, Iterator 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));      } } |

The reducer collects the datagram as a pair <word, word count> of word and its frequency from each data node and creates another datagram as a pair of word and its total frequency from all nodes.

**map-reduce organization**

conf.setMapperClass();  //In our case MapClass.**class**

conf.setCombinerClass();  //In our case Reduce.**class**

conf.setReducerClass(); //In our case again Reduce.**class**

Combiner and Reducer take the reduce class parameters because ultimately they are doing the same work at different levels.

**datagram pair**

conf.setOutputKeyClass(); // Since it is text in our case so Text.**class**

conf.setOutputValueClass(); // Since it is int in our case so IntWritable.**class**

**Step-4. Running WordCount**

Now you are ready to execute the [WordCount](http://cs.smith.edu/dftwiki/index.php/Hadoop_WordCount.java) example.

To run this example you should be inside the example directory of Hadoop:

Use the below syntax to execute any of the example,

|  |  |
| --- | --- |
| 1 | $hadoop jar /home/hadoop/hadoop--examples.jar  dft dft-output |

Execute the below command to run WordCount example:

|  |  |
| --- | --- |
| 1 | $hadoop jar /home/hadoop/hadoop/hadoop-1.1.12-examples.jar wordcount dft dft-output |

The output of command should appear like this:

**Step-5. Getting the final output**

Execute this Hadoop command to check the content in hadoop dfs directory

|  |  |
| --- | --- |
| 1 | $ hadoop dfs –ls |

The output may appear like below

Found x items drwxr-xr-x – hadoop supergroup 0 2010-03-16 11:36 /user/hadoop/dft drwxr-xr-x – hadoop supergroup 0 2010-03-16 11:41 /user/hadoop/dft-output.

You must see cross verify if the directory with *-output* at the end of your identifier (dft in our case) has been created or not.

**Checking the contents of output directory:**

Execute this command to check the contents of output directory

|  |  |
| --- | --- |
| 1 | $ hadoop dfs -ls dft-output |

The output should appear like below.

Found 2 items drwxr-xr-x – hadoop supergroup 0 2010-03-16 11:40 /user/hadoop/dft-output/\_logs -rw-r–r– 2 hadoop supergroup 518532 2013-10-31 10:31 /user/hadoop/dft-output/part-00000

To get the frequency count of each word we need to explore the file part-00000, using the following command to check the file contents:

|  |  |
| --- | --- |
| 1 | $ hadoop dfs -cat dft-output/part-00000 | less |

Note: Output file is created in the HDFS and not on your local storage. So if you want to copy the output file to your local storage, follow these simple steps.

|  |  |
| --- | --- |
| 1  2  3  4 | $ cd ~/dft  $ hadoop dfs -copyToLocal  $ hadoop dfs -copyToLocal dft-output/part-00000 .  $ hadoop dfs -copyToLocal dft-output/part-00000 |