PAVITHRA PRAKASH **A20359604**

TERASORT ON HADOOP/SPARK

Time taken to sort a Terabyte of data is measured in a benchmark called Terasort. Terasort is usually used to benchmark programming models like Hadoop and Spark that process Large Scale Data. In this assignment the comparisons of the results are based on the throughput obtained while performing the Terasort.

- In this programming assignment I have implemented the Terasort benchmark to sort 10 GB of dataset generated using a tool called 'Gensort' and as well as measured the benchmark on 100GB dataset.
- The experiment is performed on a single node as well as a cluster of 16 nodes (slaves) and a master node.
- The implementation is done using Java Programming Language and is termed as Shared Memory Sort.
- The assignment also includes performing the Terasort using Hadoop and Spark.
- The Hadoop and Spark implementation is also performed on single and 16 nodes.
- All the experiments are conducted on EC2 Instances of Amazon –c3.large.

MANUAL

The Source Code Folder includes SharedMemory Folder, Hadoop Folder.

SharedMemory folder contains the following java files and a script

SharedMem.java

smt2.java

smt4.java

smt8.java

SharedMemory.sh

Hadoop Folder contains following

hadoopsn.java

build.xml

Hadoop.jar is an executable jar to run the Hadoop program implemented in java which must be created.

Hadoop Configuration must be done prior n order to run the experiments.

Step1

Raid volume onto the instance by following the commands given below under Raid 0 commands section.

Once raided generate the 1gbdata and 10 GB data on the raided volume (say 100GB for 10GB experiments and 1000GB for 100GB experiments) using Gensort.

Step2

Unzip the Gensort after installing and navigate to 64 folder and type below commands

gensort –a 10000000 /mnt/raid/1gbdata

gensort –a 100000000 /mnt/raid/10gbdata

gensort –a 1000000000 /mnt/raid/100gbdata

Shared Memory Execution

Unzip the Prog2_Prakash_Pavithra.zip on the instance.

Navigate to SharedMemory folder using the command

Cd SourceCode/SharedMemory

To run these java files run the following Script (SharedMemory.sh)

./SharedMemory.sh

Make sure the java files have the right path of the input file before running the script.

/mnt/raid/1gbdata for 1GB Experiment

/mnt/raid/10gbdata for 10GB Experiment

/mnt/raid/100gbdata for 100GB Experiment

When the dataset size is changed one must enter the filename with path in the java file manually

Ex: When changing from 10GB dataset to 100GB dataset, open the SharedMemory.java file and change from /mnt/raid/10gbdata to /mnt/raid/100gbdata.

The new path entered must exist and should be valid.

Raid0 Commands (Step1)

The volume should be attached to the instance while creating the instance.

Navigate to Super User and execute following commands – navigating to super user is done using sudo su command

sudo fdisk –l

sudo mdadm --create --verbose /dev/md0 --level=0 --name=MY_RAID --raid-devices=number_of_volumes device_name1 device_name2

(fdisk –l will show the volume added during the creation and the device name say /dev/xvdb etc needs to be entered in the above command in place of device_name)

sudo mkfs.ext4 -L MY_RAID /dev/md0

sudo mkdir -p /mnt/raid

sudo mount LABEL=MY_RAID /mnt/raid

chmod 777 /mnt/raid

exit

Once the raid is done the dataset needs to be generated using the /mnt/raid/ path for the file path in gensort command(Step 2)

Hadoop Execution

Navigate to Hadoop folder using the command

Cd SourceCode/Hadoop

Move the hadoop.jar (Example) to the home directory using the command

mv hadoop.jar /home/

The Hadoop jar has to be saved in the home directory for running the above command and the dataset should be the name of the input data and has to be stored in hdfs.

Moving the data from /mnt/raid to hdfs:/ following command must be run

hadoop fs –put /mnt/raid/filename hdfs://

The Hadoop configuration should be complete before running the above command.

Execute the jar using the following command example

COMMAND: hadoop jar hadoop.jar hdfs:///dataset hdfs:///output

For Single Node

The command needs to be done once the Hadoop is configured on the instance

For 16 Node Cluster

Master node has to be started and configured as explained below (Configuration details) and using the image of the master node slaves needs to be created

The two config files needs to be changed slaves and core-site.xml where the IP of the slaves (current node is added in the place where IP address is present in both the files)

Now masters file needs to be created in the /hadoop/etc/Hadoop folder and should contain first its own IP address following the IP address of the slaves.

Configuration Details

In order to configure Hadoop one should change the following Files

core-site.xml

mapred-site.xml

hdfs-site.xml

yarn-site.xml

hadoop-env.sh

slaves

masters (Only in master node)

Once Hadoop is installed on the instance and the unzipped(hadoop), navigate to hadoop/etc/hadoop

Then change the above files by using vi filename command.

Now navigate to hadoop/bin and run the following command ./hdfs namenode –format.

Now enter eval `ssh-agent -s` and ssh-add HadoopUbuntu.pem (HadoopUbuntu.pem is the key used to connect to the instance).

Now navigate to hadoop/sbin and type the following command ./start-dfs.sh and when the prompt returns type ./start-yarn.sh.

Now once the prompt returns run the jps command.

If all the files are configured properly the jps will return Namenode, Datanode, Resource Manager, Node Manager and the Jps.

Now navigate to home and run the command to execute the jar (mentioned above).

Spark Execution

Once the spark is installed and unzipped add the access key and the secret key in the instance by using the following commands.

Cluster Setup

(The key can be generated in the AWS console under the security credentials where a new Access key has to be requested.)

export AWS_ACCESS_KEY_ID= enter your access key here

export AWS_SECRET_ACCESS_KEY= enter your secret key here

navigate to spark ec2 folder and run the following command script.

./spark-ec2 --key-pair=SparkUbuntu --identity-file=/home/Ubuntu/SparkUbuntu.pem --region=us-east-1 --slaves=16 --instance-type=c3.large --ebs-vol-size=1000 --ami=ami-877142ed --spot-price=0.02 launch spark

Where SparkUbuntu is the key used to connect to the instance. For identity file the path of the key file needs to be mentioned. The –slaves number mentions the nodes of the cluster one wants to create.

For 16 nodes cluster mention 16 for a single node cluster enter 1 here.

Volume can be added here and this can be raided using the commands below.

Spark at the end of the command is the name of the cluster.

Once the above command is run the cluster will be created and then the spark-shell needs to be started and code must be pasted there.

In order to start the spark shell navigate to bin folder and then type ./spark-shell

In order to paste the code type :paste

Paste the code and press ctrl-D

Code

10 GB Run

```
val starttime = System.nanoTime

val filecontent = sc.textFile("hdfs:///10gbdata")

val tosort = filecontent.map(part =>
(part.substring(0,10),part.substring(10,part.length()))).sortByKey(true,10).saveAsTextFile("10gbsoutput"
)

val stoptime = System.nanoTime

val timetaken = stoptime-starttime
```

Code

100 GB Run

```
val starttime = System.nanoTime

val filecontent = sc.textFile("hdfs:///100gbdata")

val tosort = filecontent.map(part =>
(part.substring(0,10),part.substring(10,part.length()))).sortByKey(true,10).saveAsTextFile("100gbsoutput
")

val stoptime = System.nanoTime

val timetaken = stoptime-starttime
```

Configuration Changes Details

The Config folder under the source code folder contains all the config files whose content has to be pasted in the in the instance's Config file. Navigation and commands to run is already explained.

* In the Core-site.xml the public DNS of the instance is added as a property and the temp directory where the mounted volume can reside.

* In the hdfs-site.xml the hdfs replication criteria is mentioned, the number of times the master must copy the data in its slaves is mentioned here and dfs.replication property and in my experiments I have given 1 in order to run the experiments efficiently to obtain optimal performance.

* In the mapred-site.xml all the settings needed to configure mapreduce framework is mentioned, the yarn framework is added as a property and the limit on the number of mappers and reducers needed in order to run the experiment is added as a property.

* In the yarn-site.xml the framework is configured where the resource manager is added by mentioning the address of the instance. The properties related to resource manager are specified. The classpath for the map reduce is set.

* In the hadoop-env.sh the Java home is set and the Hadoop home is set. The error handling is done over here for the heap space error etc.

* Slaves file should include the public DNS(IP address) of the instance

* Masters file in the master should include all the public DNS of the slaves and first public DNS should be of its own.