CS553 Programming Assignment #2

TeraSort on Hadoop/Spark

Instructions:

- ***Due date: 11:59PM on Monday, 03/21/16***
- ***Maximum Points: 100%***
- ***Maximum Extra Credit Points: 10%***
- *This programming assignment must be done individually.*
- *Please post your questions to the Piazza forum.*
- *Only a softcopy submission is required; it must be submitted to "Digital Drop Box" on Blackboard.*
- *For all programming assignments, please submit just the softcopy; please zip all files (report, source code, compilation scripts, and documentation) and submit it to BB.*
- *Name your file as this rule: "PROG#_LASTNAME_FIRSTNAME.{zip|tar|pdf}". E.g. "Prog2_Raicu_Ioan.tar".*
- *Late submission will be penalized at 10% per day (beyond the 7-day late pass).*

1. Introduction

The goal of this programming assignment is to enable you to gain experience programming with:

- Amazon Web Services, specifically the EC2 cloud (http://aws.amazon.com/ec2/)
- The Hadoop framework (http://hadoop.apache.org/)
- The Spark framework (http://spark.apache.org/)

In PA1, you were supposed to sign up for an account on Amazon Web Services. Through the AWSEducate online system, you should have received $35 credit to spend on Amazon AWS. IIT has just been approved as a institution member, and therefore your $35 credit should be updated to $100. If you have not received at least $35 credit this semester for you to use on PA2 and upcoming PA3, please send email to the TAs at cs553-s16@datasys.cs.iit.edu.

2. Your Assignment

This programming assignment covers the TeraSort application implemented in 3 different ways: Java, Hadoop, and Spark. Your sorting application could read a large file and sort it in place (your program should be able to sort larger than memory files). You will create 2 datasets, a small and a large dataset, which you will use to benchmark the 3 approaches to sorting: 1GB dataset and 1TB dataset. You must generate a 1TB dataset using the file generator at [8]; since storing 1TB dataset on Amazon S3 would be both expensive and slow, you are encouraged to generate the 1TB dataset on demand every time you have to perform the TeraSort application.

This assignment will be broken down into eight parts:

1) **Virtual Cluster (1-node):** Setup virtual cluster of 1 node on Amazon EC2 using d2.xlarge instance
2) **Shared-Memory TeraSort [10 points]:** Implement the Shared-Memory TeraSort application in your favorite language (without using Hadoop or Spark) and measure its performance on 1 node on a d2.xlarge instance (from here on, this will be called Share-Memory TeraSort); you should make your

Shared-Memory TeraSort multi-threaded to take advantage of multiple cores; measure the time to sort both the 1GB dataset and the 1TB dataset

3) **Virtual Cluster (16-nodes):** Setup virtual cluster of 16 nodes on Amazon EC2 (using the same AMI from step #1)
4) **Hadoop:** Install Hadoop on 16 nodes (including the HDFS distributed file system); turn off replication in order to have lower storage requirement; you must setup your own Hadoop cluster, and cannot use the Amazon Elastic MapReduce (EMR) available from Amazon
5) **Hadoop TeraSort [25 points]:** Implement the Hadoop TeraSort application, and evaluate its performance on 1 node and 16 nodes. You should be doing strong scaling experiments (keep the dataset fixed) as you scale up from 1 node to 16 nodes
6) **Spark:** Install Spark on 16 nodes; you must setup your own Spark cluster, and cannot use EMR to launch the Spark cluster
7) **Spark TeraSort [25 points]:** Implement the Spark TeraSort application, and evaluate its performance on 1 node and 16 nodes. You should be doing strong scaling experiments as you scale up from 1 node to 16 nodes.
8) **Performance [40 points]:** Compare the performance of the three versions of TeraSort (Shared-Memory, Hadoop, and Spark) on 1 node scale and explain your observations; compare the Shared-Memory performance of 1 node to Hadoop and Spark TeraSort at 16 node scales and explain your observations. Do this for both 1GB dataset and 1TB dataset.
9) **Sort with MPI [Extra Credit: 10 points]**: Implement and evaluate sorting with MPI; you must conduct the same experiments on 1GB and 1TB datasets with the same instance type.

2.1. Virtual Cluster (1-node)

Install your favorite Linux distribution in a virtual machine on Amazon EC2. If you want to use a pre-created image from Amazon, that is fine as long as you specify what image you used. Make sure ssh is enabled on your Linux install. You will need to setup several applications, such as Java [3], ANT [2] (works great as a Makefile for Java programs), Hadoop [4,5], and Spark [6,7]. You should use "spot instances" as they are less expensive, as well as "d2.xlarge" instance types (see http://aws.amazon.com/ec2/instance-types/ for more details; also http://aws.amazon.com/ec2/pricing/ for pricing). The on-demand prices for these instances is $0.69 an hour, but with spot instances, you can get them for around $0.1 to $0.15 an hour. Since you have multiple disks per instance (3 disks of 2TB each), you may find it useful to combine the disks into a RAID-0 to give you the best performance possible.

2.2. Shared-Memory TeraSort [10 points]

TeraSort [1] is an application which sorts a file-resident dataset; it should be able to sort datasets that are larger than memory. You are to implement the TeraSort application in your favorite language, without MapReduce/Hadoop or Spark. TeraSort should be multi-threaded (to process large files concurrently), and should only run on 1 virtual node. You can develop your code on any machine, but your performance evaluation must be done on Amazon EC2 on a "d2.xlarge" instance. Measure the time to execute the TeraSort application on two datasets (a 1GB and 1TB dataset produced with the tool XXX) on 1 node. Vary the number of threads from 1 to 8, to find the best performance. Save the first 10 lines of output from the Shared-Memory TeraSort application, as well as the last 10 lines of output, for each dataset, in terasort-shared-memory-1GB.txt and terasort-shared-memory-1TB.txt respectively.

2.3. Virtual Cluster (17-nodes)

Setup virtual cluster of 17 nodes (you may find it useful to have 1 node for control, and 16 nodes for compute/storage) on Amazon EC2 (using the same AMI from step #1). You may find useful (as you scale up to multiple VMs) to install a shared file system (NFS) between the virtual machines, but it is not required. The shared file system can be used to distribute binaries, application installations, and configuration files. Alternatively, you may find useful to install S3FS to convert S3 into a POSIX-compatible shared filesystem.

2.4. Hadoop

Install Hadoop including the HDFS distributed file system. You may want to configure Hadoop to run the job tracker and filesystem metadata service on separate nodes, leaving 16 nodes available to run workers for map and reduce tasks. Please follow instructions at [4,5] on how to install Hadoop. You can turn off data replication on the HDFS file system to ensure you get the most performance out of your system.

2.5. Hadoop TeraSort [25 points]

TeraSort fits very well with the MapReduce programming model making it a great example to understand the Hadoop Map/Reduce programming style.

After you run Hadoop on 1 node first, you will likely have to modify these configuration files as you go to a multi-node run:

1) conf/master
2) conf/slaves
3) conf/core-site.xml
4) conf/hdfs-site.xml
5) conf/mapred-site.xml

You are to write a description of the function of each file, and what modifications you had to make to go from 1 node to multiple nodes. Please answer the following questions:

1) What is a Master node? What is a Slaves node?
2) Why do we need to set unique available ports to those configuration files on a shared environment? What errors or side-effects will show if we use same port number for each user?
3) How can we change the number of mappers and reducers from the configuration file?

Save the first 10 lines of output from the Hadoop TeraSort application, as well as the last 10 lines of output, for each dataset, in terasort-hadoop-1GB.txt and terasort-hadoop-1TB.txt respectively.

2.6. Spark

Install Spark on 17 nodes, and configure it to run the HDFS file system in order to be able to handle the 1TB dataset. You can turn off data replication on the HDFS file system to ensure you get the most performance out of your system. Also, make sure to configure your RDD storage in such a way that you do not create replicas of your data; this will also give you the best performance at the cost of resilience in case of failures.

2.7. Spark TeraSort [25 points]

Implement the Spark TeraSort application. Save the first 10 lines of output from the Spark TeraSort application, as well as the last 10 lines of output, for each dataset, in terasort-spark-1GB.txt and terasort-spark-1TB.txt respectively.

2.8. Performance [40 points]

Compare the performance of the three versions of TeraSort (Shared-Memory, Hadoop, and Spark) on 1 node scale and explain your observations; compare the Shared-Memory performance of 1 node to Hadoop and Spark TeraSort at 16 node scales and explain your observations. You should be doing strong scaling experiments as you scale up from 1 node to 16 nodes. You only need to do two different scales, 1 node and 16 nodes (no need to incrementally do 1, 2, 4, 8, and 16).

Hint: Since your d2.xlarge instance will have 4 virtual cores, you will configure Spark and Hadoop to run 4 workers, mappers or reducers. For example, for Hadoop, you can change the total amount of mapper and reducer by editing the configuration files "conf/mapred-site_template.xml"and "conf/mapred-site.xml".

Draw an execution line chart and a speed up line chart for 1 node and 16 node cases, for Java, Hadoop, and Spark TeraSort. For Spark and Hadoop, compute two different speedups, using different base cases; one speedup (speedup-shared-memory) should be relative to the Shared-memory TeraSort performance (note that you might get a speedup less than 1); the second speedup (speedup-spark & speedup-hadoop) should be relative to the Spark or Hadoop performance at 1 node scale respectively (should be a number greater than 1).

What conclusions can you draw? Which seems to be best at 1 node scale? How about 16 nodes? Can you predict which would be best at 100 node scale? How about 1000 node scales? Compare your results with those from the Sort Benchmark [9], specifically the winners in 2013 and 2014 who used Hadoop and Spark. Also, what can you learn from the CloudSort benchmark, a report can be found at [10]. All of these questions must be addressed in your final report write-up for this assignment.

2.9. MPI TeraSort [Extra credit: 10 points]

Implement and evaluate sorting across with MPI (MPI TeraSort). Your sorting application could read a large file and sort it in place (your program should be able to sort larger than memory files). Use the file generator at [8] to generate a 1GB and 1TB dataset. For MPI TeraSort, you will have to setup MPI on your virtual cluster of 16 nodes, and implement MPI TeraSort. Measure the performance of the MPI TeraSort at 1 node and 16 node scales. Compare your MPI TeraSort with Spark and Hadoop at 1 node and 16 node scales. Save the first 10 lines of output from the MPI TeraSort application, as well as the last 10 lines of output, for each dataset, in terasort-mpi-1GB.txt and terasort-mpi-1TB.txt respectively.

3. Deliverables

You are to write a report (prog2_report.pdf). Add a brief description of the problem, methodology, and runtime environment settings. Discuss the installation steps you took to setup your virtual cluster. Please include any difficulties you might have incurred in your setup of the virtual cluster. Also, include a detailed description of what OS you used (Linux distribution, kernel), what ANT version, what Java version, what Hadoop version, what Spark version, and what MPI version you used.

You are to draw graphs showing execution time data and speedup data. Please explain your results. Please include a comparison between the SharedMemory TeraSort compared to the Hadoop TeraSort and Spark TeraSort for 1 node and Hadoop and Spark at 16 nodes, and potentially the MPI TeraSort. Can you explain the difference in performance?

You are required to turn in a zipped or tarred package named as "prog2_lastname1_firstname1.{zip|tar}" on BlackBoard.

1) Source code and configuration files for both the single node and multiple node cases; do not include the dataset files
2) Scripts used to automate the deployment of your virtual cluster
3) A readme.txt file with explanations of code, which file contains what, and the commands needed to compile and execute the different scenarios

4) Output files “terasort-shared-memory-1GB.txt”, “terasort-shared-memory-1TB.txt”, “terasort-hadoop-1GB.txt”, “terasort-hadoop-1TB.txt”, “terasort-spark-1GB.txt”, “terasort-spark-1TB.txt”, “terasort-mpi-1GB.txt”, “terasort-mpi-1TB.txt”,
5) Report file “prog2_report.pdf”

4. References

[1] https://hadoop.apache.org/docs/r2.7.1/api/org/apache/hadoop/examples/terasort/package-summary.html

[2] http://ant.apache.org/bindownload.cgi

[3] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[4] http://hadoop.apache.org/docs/current1/mapred_tutorial.html

[5] http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/ClusterSetup.html

[6] http://spark.apache.org/downloads.html

[7] http://spark.apache.org/docs/latest/cluster-overview.html

[8] http://www.ordinal.com/gensort.html

[9] http://sortbenchmark.org

[10] http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf