# CS553 Programming Assignment #2

## TeraSort on Hadoop/Spark

***Instructions:***

- ***Due date: 11:59PM on Monday, 11/27/17***
- ***Maximum Points: 100%***
- ***Maximum Extra Credit Points: 20%***
- *This programming assignment can be done in groups of up to 3 students*
- *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#_LASTNAME1_LASTNAME2_LASTNAME3.tgz". E.g. "Prog2_Raicu_Peng_Mathew.tgz".*
- *Late submission will be penalized at 20% per day (beyond the 4-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 will received $100 credit to spend on Amazon AWS. More information on how to get you AWS credit will come soon.

### 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, also known as external sort). You will create 2 datasets, a small and a large dataset, which you will use to benchmark the 3 approaches to sorting: 128GB dataset and 1TB dataset. You must generate your datasets 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 want to benchmark your system.

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 gcc, 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 "i3.large" and "i3.4xlarge" 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 $0.156 and $1.248 an hour, but with spot instances, you can get them for much cheaper. Since you may have

multiple disks per instance, you may find it useful to combine the disks into a RAID-0 to give you the best performance possible.

This assignment will be broken down into several parts, as outlined below:

1) **Virtual Cluster (1-node i3.large):** Setup virtual cluster of 1 node on Amazon EC2 using i3.large instance; generate a dataset of 128GB in size; during performance evaluation, do not include measurement of the time to generate the data, load the data, or verify that the data is sorted correctly; performance evaluation should only include the time to sort the data; after experiments (a), (b), and (c), terminate the instance
    a. **Shared-Memory TeraSort:** Implement the Shared-Memory TeraSort application in your favorite language (without using Hadoop or Spark); generate the 128GB dataset and measure the time to sort it; you should make your Shared-Memory TeraSort multi-threaded to take advantage of multiple cores and SSD storage (which also requires multiple concurrent requests to achieve peak performance)
    b. **Hadoop TeraSort:** Install Hadoop (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; all Hadoop components should be configured on this 1 node; load the 128GB dataset into HDFS; implement the Hadoop TeraSort application, and evaluate its performance on 1 node
    c. **Spark TeraSort:** Install Spark (including the HDFS distributed file system); you must setup your own Spark cluster, and cannot use EMR to launch the Spark cluster; load the 128GB dataset into HDFS (unless its already loaded); implement the Spark TeraSort application, and evaluate its performance on 1 node
    d. **MPI TeraSort:** Implement and evaluate external sort with MPI; generate the 128GB dataset and measure the time to sort it; you should make your MPI TeraSort uses multiple processes (ranks) to take advantage of multiple cores and SSD storage (which also requires multiple concurrent requests to achieve peak performance)
2) **Virtual Cluster (1-node i3.4xlarge):** Setup virtual cluster of 1 node on Amazon EC2 using i3.4xlarge instance; generate a dataset of 1TB in size; during performance evaluation, do not include measurement of the time to generate the data, load the data, or verify that the data is sorted correctly; performance evaluation should only include the time to sort the data; after experiments (a), (b), and (c), terminate the instance
    a. **Shared-Memory TeraSort:** Evaluate Shared-Memory TeraSort application and measure the time to sort the 1TB dataset on 1 node (don't forget to increase the number of threads to make use of the 16-core instance)
    b. **Hadoop TeraSort:** Configure HDFS with no replication; evaluate Hadoop TeraSort application and measure the time to sort the 1TB dataset on 1 node (don't forget to increase the number of mappers and reducers to make use of the 16-core instance)
    c. **Spark TeraSort:** Configure HDFS with no replication; evaluate Spark TeraSort application and measure the time to sort the 1TB dataset on 1 node (don't forget to increase the parallelism to make use of the 16-core instance)
    d. **MPI TeraSort:** Evaluate MPI TeraSort application and measure the time to sort the 1TB dataset on 1 node (don't forget to increase the number of processes to make use of the 16-core instance)
3) **Virtual Cluster (8-nodes i3.large):** Setup virtual cluster of 8 nodes on Amazon EC2 using i3.large instance types; generate a dataset of 1TB in size; during performance evaluation, do not include measurement of the time to generate the data, load the data, or verify that the data is sorted correctly; performance evaluation should only include the time to sort the data; after experiments (a) and (b), terminate the instance

a. **Hadoop TeraSort:** Configure Hadoop to span all 8 nodes, and HDFS with no replication; evaluate Hadoop TeraSort application and measure the time to sort the 1TB dataset
b. **Spark TeraSort:** Configure Spark and HDFS to span all 8 nodes, and HDFS with no replication; make sure to disable replication for RDD as well; evaluate Spark TeraSort application and measure the time to sort the 1TB dataset
c. **MPI TeraSort:** Configure MPI to span all 8 nodes; configure a shared/parallel/distributed file system (e.g. NFS, Lustre, OrangeFS, Gluster, Ceph) without replication; evaluate MPI TeraSort application and measure the time to sort the 1TB dataset on 8 nodes; you could probably configure MPI to work without a shared file system, but the implementation of your application might be more complex; note that NFS will likely give the worst performance due to the poor scalability of data access performance, so you are encouraged to use a more sophisticated distributed/parallel file system instead

4) **Performance:** Compare the performance of the four versions of TeraSort (Shared-Memory, Hadoop, Spark, and MPI) on 1 node scale and 8 node scale with both the 128GB and 1TB dataset; fill in the table below, and then derive new tables or figures (if needed) to explain the results. Your time should be reported in seconds. Some of the things that will be interesting to explain are: how many threads, mappers, reducers, you used in each experiment; how many times did you have to read and write the dataset for each experiment; what speedup and efficiency did you achieve (use weak scaling)? Complete Table 1 outlined below.

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. Only use NFS or S3FS for configuration and application binaries, as they will likely be slow to process large datasets. Since your instances will have 2 to 16 virtual cores, you will configure Spark and Hadoop to run multiple 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".

Compare the performance of the three versions of TeraSort (Shared-Memory, Hadoop, Spark, and MPI [EC]) on 1 node scale on two different types of instances as well as on a virtual cluster of 8 nodes, and explain your observations; compare the Shared-Memory performance of 1 node to Hadoop and Spark TeraSort at 8 node scales and explain your observations. You should be doing weak scaling experiments as you scale up from 1 node to 8 nodes. You only need to do two different scales, 1 node and 8 nodes (no need to incrementally do 1, 2, 4, and 8).

What conclusions can you draw? Which seems to be best at 1 node scale? How about 8 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.

Perform the experiments outlined above, and complete the following table:

*Table 1: Performance evaluation of TeraSort*

| **Experiment (instance/dataset)** | **Shared Memory TeraSort** | **Hadoop TeraSort** | **Spark TeraSort** | **MPI TeraSort** |
|---|---|---|---|---|
| **Compute Time (sec) [1xi3.large 128GB]** | | | | |
| **Data Read (GB) [1xi3.large 128GB]** | | | | |

| | | | | |
|---|---|---|---|---|
| **Data Write (GB) [1xi3.large 128GB]** | | | | |
| **I/O Throughput (MB/sec) [1xi3.large 128GB]** | | | | |
| **Compute Time (sec) [1xi3.4xlarge 1TB]** | | | | |
| **Data Read (GB) [1xi3.4xlarge 1TB]** | | | | |
| **Data Write (GB) [1xi3.4xlarge 1TB]** | | | | |
| **I/O Throughput (MB/sec) [1xi3.4xlarge 1TB]** | | | | |
| **Compute Time (sec) [8xi3.large 1TB]** | N/A | | | |
| **Data Read (GB) [8xi3.large 1TB]** | N/A | | | |
| **Data Write (GB) [8xi3.large 1TB]** | N/A | | | |
| **I/O Throughput (MB/sec) [8xi3.large 1TB]** | N/A | | | |
| **Speedup (weak scale)** | | | | |
| **Efficiency (weak scale)** | | | | |

## 3. Grading

The grading will be done according to the rubric below:

- Virtual cluster setup with automation scripts: 10 points
- Required: Shared memory terasort implementation/scripts: 30 points
- Hadoop Terasort implementation/scripts: 10 points
- Spark Terasort implementation/scripts: 10 points
- MPI Terasort implementation/scripts: 10 points
- Readme.txt: 5 points
- Performance evaluation, data, explanations, etc: 15 points per system (for a maximum of 60 points)
- Followed instructions on deliverables: 5 points

The maximum score that will be allowed is 100 points, plus 20 points extra credit. Note that if the entire assignment is completed perfectly, the maximum score would be 140 points, but only 120 points will be given.

## 4. Deliverables

You are to write a report (pa2_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 fill in the table on the previous page. Please explain your results, and explain the difference in performance?

You are required to turn in a zipped or tarred package named as "pa2_lastname1_lastname2.tgz" on BlackBoard.

1) Source code and configuration files for both the single node and multiple node cases for all 4 versions of the code (shared memory, Hadoop, Spark, and MPI); 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) Screenshots:

a) Include screenshots of running instances on amazon EC2 with your name and instance IPs clearly visible; include separate screenshots for shared memory, Hadoop, Spark, and MPI
b) Include screenshots of the completion of the sort invocations with clear timing information and experiment details; include separate screenshots for shared memory, Hadoop, Spark, and MPI
c) Include screenshot of valsort [8] execution after sorting the datasets for each sort invocation; include separate screenshots for shared memory, Hadoop, Spark, and MPI

5) Report file "prog2_report.pdf"

## 5. 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