CS553 Homework #6

Sort on Hadoop/Spark

Instructions:

- *Assigned date: Tuesday November 1st, 2022*
- *Due date: 11:59PM on Friday November 11th, 2022*
- *Maximum Points: 100%*
- *This homework can be done in teams of up to 3 students*
- *Please post your questions to BB*
- *Only a softcopy submission is required; submission is a 2-step process: 1) push changes to GIT repository, and email confirmation will be sent to your HAWK email address at the deadline; a confirmation document with all team member names and A# must be submitted through BlackBoard for your submission to be graded; only 1 student must submit the assignment, and only the submitting student will receive the confirmation email*
- *Late submission will be penalized at 10% per day (students working alone can get 1 late-day submission without penalty)*

1. Introduction

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

- The Hadoop framework (http://hadoop.apache.org/)
- The Spark framework (http://spark.apache.org/)

In Homework #5, you implemented an external sort and compared it to the Linux sort. You will now expand into implementing sort with Hadoop and with Spark.

2. Your Assignment

This programming assignment covers sort through Hadoop and Spark on multiple nodes. You must use a Chameleon node using Bare Metal Provisioning (https://www.chameleoncloud.org). You must deploy Ubuntu Linux 22.04 using "compute-haswell" nodes, at the IIT sites. Once you create a lease (up to 7 days are allowed), and start your 1 physical node, and Linux boots, you will find yourself with a physical node with 24 CPU cores, 48 hardware threads, 128GB of memory, and 250GB SSD hard drive. You will install your favorite virtualization tools (e.g. virtualbox, LXD/KVM, qemu), and use it to deploy two different type of VMs with the following sizes: tiny.instance (4-cores, 8GB ram, 20GB disk), small.instance (4-cores, 8GB ram, 45GB disk), and large.instance (16-cores, 32GB ram, 180GB disk).

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

Hadoop File System and Hadoop Install: Download, install, configure, and start the HDFS system (that is part of Hadoop, https://hadoop.apache.org) on a virtual cluster with 1 large.instance + 1 tiny.instance, and then again on a virtual cluster with 4 small.instances + 1 tiny.instance. You must set replication to 2 (instead of the default 3), or you won't have enough storage capacity to conduct your experiments on the 24GB dataset.

Datasets: Once HDFS is operational, you must generate your dataset with gensort (http://www.ordinal.com/gensort.html); you will create 4 workloads: data-3GB, data-6GB, data-12GB, and data-24GB. You may not have enough room to store them all, and run your compute workloads. Make sure to cleanup after each run. Remember that you will typically need 6X the storage, as you have the original input data (2x),

temporary data (2x), and output data (2x). Configure Hadoop to run on the virtual cluster, on 1 large.instance + 1 tiny.instance as well as the separate installation on 4 small.instances + 1 tiny.instance. The tiny.instance will run parts of Hadoop (e.g. name node, scheduler, etc).

Spark Install: Download, install, configure, and start Spark (https://spark.apache.org). Note that you will need the HDFS installation for Hadoop to work from and to.

Linux Sort: Run the Linux Sort on the small.instance and large.instance defined above on the different datasets.

Hadoop Sort: Implement the HadoopSort application (you must use Java). You must specify the number of reducers to ensure you use all the resources of the 4-node cluster. You must read the data from HDFS, sort it, and then store the sorted data in HDFS and validate it with valsort. Measure the time from reading from HDFS, sorting, and writing to HDFS; do not include the time to run valsort.

Spark Sort: Implement the SparkSort application (you must use Java). Make sure to use RDD to speed up the sort through in-memory computing. You must read the data from HDFS, make use of RDD, sort, and write the sorted data back to HDFS, and finally validate the sorted data with valsort. Measure the time from reading from HDFS, sorting, and writing to HDFS; do not include the time to run valsort.

Performance: Compare the performance of your shared-memory external sort, the Linux "sort" (more information at http://man7.org/linux/man-pages/man1/sort.1.html), Hadoop Sort, and Spark Sort on a 4-node cluster with the 3GB, 6GB, 12GB, and 24GB datasets. Fill in the table below, and then derive new tables or figures (if needed) to explain the results. Your time should be reported in milliseconds.

Complete Table 1 outlined below. Perform the experiments outlined above, and complete the following table:

Table 1: Performance evaluation of sort (report time to sort in milliseconds); each instance below needs a tiny.instance for the name node

Experiment	Linux	Hadoop Sort	Spark Sort
1 small.instance, 3GB *dataset*			
1 small.instance, 6GB *dataset*			
1 small.instance, 12GB *dataset*			
1 large.instance, 3GB *dataset*			
1 large.instance, 6GB *dataset*			
1 large.instance, 12GB *dataset*			
1 large.instance, 24GB *dataset*			
4 small.instances, 3GB *dataset*	N/A		
4 small.instances, 6GB *dataset*	N/A		
4 small.instances, 12GB *dataset*	N/A		
4 small.instances, 24GB *dataset*	N/A		

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?

For the 24GB workload, monitor the disk I/O speed (in MB/sec), memory utilization (GB), and processor utilization (%) as a function of time, and generate a plot for the entire experiment. Here is an example of a plot that has cpu utilization and memory utilization (https://i.stack.imgur.com/dmYAB.png), plot a similar looking graph but with the disk I/O data as well as a 3rd line. Do this for both shared memory benchmark (your code) and for the Linux Sort. You might find some online info useful on how to monitor this type of information

(https://unix.stackexchange.com/questions/554/how-to-monitor-cpu-memory-usage-of-a-single-process). For multiple instances, you will need to combine your monitor data to get an aggregate view of resource usage. Do this for all four versions of your sort. After you have all six graphs (2 system configurations and 3 different sort techniques), discuss the differences you see, which might explain the difference in performance you get between the two implementations. Make sure your data is not cached in the OS memory before you run your experiments.

Note that you do not have to artificially limit the amount of memory your sort can use as the VMs will be configured with a limited amount of memory, although you may still want to limit your memory usage based on available memory. What conclusions can you draw? Which seems to be best at 1 node scale (1 large.instance)? Is there a difference between 1 small.instance and 1 large.instance? How about 4 nodes (4 small.instance)? What speedup do you achieve with strong scaling between 1 to 4 nodes? What speedup do you achieve with weak scaling between 1 to 4 nodes? How many small.instances do you need with Hadoop to achieve the same level of performance as your shared memory sort? How about how many small.instances do you need with Spark to achieve the same level of performance as you did with your shared memory sort?

Can you predict which would be best if you had 100 small.instances? How about 1000? Compare your results with those from the Sort Benchmark (http://sortbenchmark.org), 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 (http://sortbenchmark.org/2014_06_CloudSort_v_0_4.pdf).

3. What you will submit

The grading will be done according to the rubric below:

- Hadoop (installation/config) sort implementation/scripts: 20 points
- Spark (installation/config) sort implementation/scripts: 20 points
- Performance evaluation (data): 30 points
- Performance evaluation (Q&A and explanations): 30 points

The maximum score that will be allowed is 100 points.

You are to write a report (hw6_report.pdf). Add a brief description of the problem, methodology, and runtime environment settings. You are to fill in the table on the previous page. Please explain your results, and explain the difference in performance? Include logs from your application as well as valsort (e.g. standard output) that clearly shows the completion of the sort invocations with clear timing information and experiment details; include separate logs for Linux sort, Hadoop sort, and Spark sort, for the 24GB dataset. Valsort can be found as part of the gensort suite (http://www.ordinal.com/gensort.html), and it is used to validate the sort. As part of your submission you need to upload to your private git repository your run scripts, build scripts, the source code for your implementation (Hadoop sort and spark sort), configuration files for both Hadoop and Spark, the report, a readme file (with how to build and use your code), and several log files. Make sure to answer all the questions posed from Section 2. Some of your answers might require a graph or table with data to substantiate your answer. Here are the specific files you need to submit:

- build.xml (Ant) / pom.xml (Maven)
- HadoopSort.java
- SparkSort.java
- Scripts
- hw6_report.pdf
- readme.txt
- linsort24GB.log
- hadoopsort24GB.log
- sparksort24GB.log
- core-site.xml (Hadoop config file)
- hdfs-site.xml (Hadoop config file)
- yarn-site.xml (Hadoop config file)
- mapred-site.xml (Hadoop config file)
- spark-env.sh (Spark environment variables)

When you have finished implementing the complete assignment as described above, you should submit your solution to your private git repository. Each program must work correctly and be detailed in-line documented.

Submit code/report through GIT. To submit your work, simply commit your PDF file and push your work to Github. You will have to submit your solution to a private git repository created for you at git@github.com:datasys-classrooms/cs553-fall2022-hw6-<team name>.git. All repositories will be collected automatically every day at midnight. The repository is created through GitHub Classroom and you will need to accept the assignment before you can clone it at https://classroom.github.com/a/Ccsb5JVp. You will also need to create a new team or join an existing team. Your submission will not be graded unless you submit a confirmation document through BlackBoard (BB) that clearly shows the pushing of your final homework to your GIT repository. This confirmation document can simply be a screen shot of your final commands to push your repository to GIT. The timestamp on the BB submission will be used to determine if the submission is on-time. ***You must also include the names and A# of all your team members in this confirmation document.*** If you cannot access your repository contact the TAs. You can find a git cheat sheet here: https://www.git-tower.com/blog/git-cheat-sheet/.

Grades for late programs will be lowered 10% per day late.