CS 6240: Assignment 4

Goal: Explore secondary sort and work with HBase to experience how MapReduce can interact with such a distributed key-value store.

This homework is to be completed individually (i.e., no teams). You must create all deliverables yourself from scratch. In particular, it is not allowed to copy someone else's code or text and modify it. (If you use publicly available code/text, you need to cite the source in your report!)

All deliverables for this HW are due as stated on canvas. For late submissions you will lose one percentage point per hour after the deadline. This HW is worth 100 points and accounts for 10% of your overall homework score. To encourage early work, you will receive a 2-point bonus if you submit your solution on or before the early submission deadline stated on canvas. (Notice that your total score cannot exceed 100 points, but the extra points would compensate for any deductions.) Please submit your solution through the canvas assignment submission system and make sure your report is a **PDF** file. Always package all your solution files, including the report, into a single ZIP file. Make sure you are using standard ZIP, i.e., the format also used by Windows archives.

You have about a week to work on this assignment. Of course, the earlier you work on this, the better.

Computation Challenge

We continue working with the flight data set from HW 3. Our goal now is to compute the pattern of monthly delays for each airline. More precisely, for **each airline**, your program should produce an output line like this: *(It does not matter which airlines are in the same part-* file and how many such part files your program produces.)*

AIR-A, (1, A1), (2, A2),…, (11, A11), (12, A12)
AIR-B, (1, B1), (2, B2),…, (11, B11), (12, B12)

Here AIR-A stands for an airline name (either the full name or some meaningful abbreviation). Pair (i, Ai) indicates the average delay Ai of airline AIR-A in month i of the year 2008. For example, (6,17) means that AIR-A had an average delay of 17 minutes in June 2008. (Similarly, the entries in the line for AIR-B indicate AIR-B's monthly average delays for 2008.) The average monthly delay should be computed from the values of attribute **ArrDelayMinutes** over all flights of that airline that departed in that month in 2008. Ignore records that are missing one of the attributes needed for this computation, e.g., delay or departed info. To avoid clutter, make sure the reported final delay numbers are rounded up to the nearest integer. For example, if you initially computed AIR-A's average June delay as 1 hour and 2.2727615 minutes, the corresponding pair should be (6, 63).

Make sure that in the output all (i, Ai) pairs for an airline are sorted in increasing order of i, i.e., the month, as shown in the example lines above.

You are expected to provide solutions using both the below stated formats.

Solution Using Secondary Sort (Value-To-Key Conversion)

Write a Hadoop MapReduce program SECONDARY that solves the computation challenge using the value-to-key conversion design pattern. Think about how this pattern can be used in the most meaningful and effective way to achieve good performance and minimize the amount and complexity of the code you write. Design your program in such a way that it can make good use of 5 to 10 machines.

Hint: If you need to access the value field that got moved into the key by the design pattern, think about how to best achieve this. Try if you can access the different keys that were assigned to the same reduce call by the grouping comparator. For example, try to iterate over the *values*, but output the *key* in each iteration. Does it change?

Solution Using HBase

Read the AWS documentation to find out how to use HBase for EMR computations. You might also want to set up HBase on your development machine to be able to test and debug your code. There are many tutorials for how to do this. (Some students found http://archanaschangale.wordpress.com/2013/08/29/installing-apache-hbase-on-ubuntu-for-standalone-mode/ and http://archanaschangale.wordpress.com/2013/09/11/connecting-hbase-using-java/ useful, but we do not endorse or promote this particular tutorial in any way.) In general, we recommend keeping things simple by using HBase in the standalone mode (not in pseudo-distributed mode).

To solve the computation challenge using HBase, write two MapReduce programs:

1. H-POPULATE: This program reads records from the input file and writes each record 1-to-1 to an HBase table. All records should be stored in the same table. For input record r, there should be exactly one matching row r' in that table. *Make sure all attributes (fields) of the record are present in the HBase table as well.* In other words, if the input file contains n records, then the HBase table should contain the corresponding n rows with all the fields from the n input records.
2. H-COMPUTE: This program reads from the HBase table to generate the desired output file. Decide how to best query HBase, what computation is needed, and where to perform query and computation (Map or Reduce).

Your program should satisfy the following requirements:

- Setting up and maintaining connections to HBase consumes resources. Hence you need to make sure that you minimize this overhead. If you access HBase many times in a single task, then the task should establish the connection once, not for every single access.

- Choose the row key wisely to be able to take advantage of HBase's built-in sorting functionality. Exploit this sorting functionality in H-COMPUTE.
- Both MapReduce programs should be designed in such a way that they make good use of 5 to 10 machines.

Hints: To use HBase efficiently, ideally each task should access a different region of the table. For example, consider a program that accesses HBase in the Map phase:

- Using a full table scan in each Map task (or, worse yet, each Map function call), would be highly inefficient. Each region server would perform a full read of its region for each Map task (!) and then send all the data. Adding a filter to the scan would reduce the data transfer, but not prevent the repeated full scan.
- Check out TableInputFormat. It automatically creates a Map task for each table region. Hence each Map task reads from a different region server, avoiding the repeated full table scan. However, think carefully how region boundaries line up or not (or could be made to line up) with the final output results you are trying to compute.
- HBase supports range queries, where you can explicitly specify a start and stop row key. This is a powerful feature to ensure that each Map task (or Map function call) reads only a small part of the table. However, to use this feature, the task using the range query needs to know which start and stop row it is responsible for. You are allowed to generate auxiliary files to achieve this.

Report

Write a brief report about your findings, using the following structure. Add screenshots to provide sufficient evidence wherever possible.

Header

This should provide information like class number, HW number, and your name.

Source Code (60 points total)

Show the source code for programs SECONDARY (30 points), H-POPULATE (15 points), and H-COMPUTE (15 points), but remove all boilerplate code. Only include those lines in the report that perform the actual "work". Stated differently, show the Java code that closely corresponds to the pseudo code.

Make sure your code is clean and well-documented. Messy and hard-to-read code will result in point loss. In addition to correctness, efficiency is also a criterion for the code grade.

Performance Comparison and Analysis (22 points total)

Run each program in Elastic MapReduce (EMR) on the entire big flight data set, using the following two configurations:

- 6 machines (1 master and 5 workers)
- 11 machines (1 master and 10 workers)

For comparison, use the same machine type for each experiment. Since HBase currently requires at least m1.large machines, use this type for each run.

Report the total running time of each program for each configuration. (6 numbers, 2 points each)

Discuss which program you believe is better by comparing (i) coding and setup effort, (ii) performance, and (iii) scalability. Try to explain reasons for performance differences in a concise manner, citing concrete evidence to make your case. (5 points)

Notice that poor load balancing can result in high running time and hence poor speedup. Make sure you think carefully about how to balance load. Briefly discuss how you designed your program to achieve good load balance and how you determined if your approach was successful. (5 points)

Deliverables

1. The report as discussed above. (1 PDF file)
2. The source code for each program. (6 points)
3. The syslog and stderr files for a successful run of each of the three programs. (6 plain text files) (12 points)