Will Hembree
CS6240
Homework 4
March 18, 2018

SECONDARY

```
Class secondary {
        Class Mapper {
                Map (key, value) {

                        isYear2008 = value.year
                        isNotCancelled = value.cancelled
                        isValidArrDelayMin = !value.arrDelayMin.equals(“”)
                        isValidMonth = !value.month.equals(“”)
                        canComputeAvgDelay = isValidArrDelayMin && isValidMonth

                        If (isYear2008 && isNotCancelled && canComputeAvgDelay) {
                                emit((uniqueCarrier, month),
                                        month, arrDelayMinutes)
                        }
                }
        }
        Class Reducer {
        HashMap taskLevelHMap
                Reduce (key, values) {
                        // key : month
                        // value : (total delay time (min), total flights)
                        HashMap inReducerHMap
                        foreach v in values:
                                month = v.month
                                arrDelayMin = v.arrDelayMinutes

                                If (inReducerHMap.containsKey(month)) {// New entry
                                updatedDelay = inReducerHMap.getArrDelayMin + arrDelayMin
                                updatedFlightTot = inReducerHMap.getMonth + 1

                                inReducerHMap.put(month, (updatedDelay, updatedFlightTot))
                                } else {// Existing entry
                                inReducerHMap.put(month, (arrDelayMinutes, 1))
                                }
                        }
                        Airline = “AIR-” + key.uniqueCarrier
```

```
                    // Helper method (see code) to turn inReducerHMap into a string of
                    // (airline1, average delay), (airline2, average delay), ...
                    avgDelaysPerMonth = getAvgDelaysPerMonth(inReducerHMap)

                    taskLevelHMap.put(airline, avgDelaysPerMonth)
        }

        Cleanup (context) {
            Foreach row in taskLevelHMap:
            emit(row.getKey(), row.getValue())
        }
}
```

My approach for secondary is to emit both the unique carrier and the month as the key. This allows me to sort by month using a customized keyComparator, partition by airline using a custom partition function, and group by unique carrier using the GroupComparator. For the values I emit both the month and the delay time. It seems redundant to have the month in both the key and the value fields, but when the group comparator is called, the month in the key can no longer be relied upon.

From the mapper, there is one reduce task per airline, i.e., each reduce task receives the delay times for the respective airline. This allows me to easily keep track of the respective delay time for the the given month. Otherwise, some complicated logic would be necessary in the reducer to keep track of the month.

Since each reducer handles a different airline, another option to keep track of the month would be to use a global counter for the airline. However, the number of airlines is difficult to anticipate, and from my reading I've learned it's best not to use more than a dozen or so global counters.

In terms of load balancing, the custom partitioner ensures even distribution of the data by computing a hash of the uniqueCarrier improving speedup and potential scaleup.

A. 6 Machines
Total Running Time ~ 2 minutes
B. 11 Machines
Total Running Time ~ 1 minute

HPOPULATE

```
Class Mapper {
        Setup () {
```

```
            New HBaseTable
        }
        Mapper (key, value) {
            Key = value.FlightDate + value.UniqeCarrier + value.Origin + value.FlightNum

            record.addRoKey(Key)
            record.add("FlightAttributes", "Year", value.year)
            record.add("FlightAttributes", "Attributes", value)

            HBaseTable.put(record)
        }
        Cleanup () {
            HBaseTable.close()
        }
}
```

Regarding the row-key choice, I used (flightDate, uniqueCarrier, origin, and flightNum) to ensure uniqueness. In addition, by using uniqueCarrier as part of the row-key, fewer region servers need to be contacted when sorting occurs (assuming approximately uniform distribution). And by making flightDate (specifically month) part of the row-key this takes advantage of the the fact that HBase scanners start at a specified row and then return the following rows in order. For each airline, the total average delay per month can easily be calculated now by increasing order of the month attribute. This choice distinctly contrasts choosing airline or month alone as part of the row key. In the former case, the same number of region servers would be used, but the month would be randomly distributed requiring further processing time. In the latter case, there would be the fewest number of region servers (12) and it would be irrelevant (assuming one regions server per month), but the airlines would be mixed across each regions and would require extra processing time to combine across regions.

The homework states that all of the attributes should be stored in HBase (*Make sure all attributes(fields) of the record are present in the HBase table as well.*). However, that seems odd considering the only field that is not in the key need for the computation is arrDelayMinutes. Performance could be improved by just storing arrDelayMinutes instead of all of the attributes in a column.

To minimize resource consumption, the HBase table is created in the setup function so that the task only creates the table once.

I also applied a filter to the scan that inspects for flight records in the desired year (2008), thus filtering out undesired data. Given the large amount of data, this will significantly reduce cost and save bandwidth.

**** I'm unable to get HPOPULATE to work on AWS. It keeps timing out. I've included the stderr and the results of a local run.****

A. 6 Machines
Total Running Time =
B. 11 Machines
Total Running Time =

**** Without HPOPULATE to seed HBase, I'm unable to run HCOMPUTE on AWS. I've included the results of a local run.****

HCOMPUTE

Pseudo-code same as SECONDARY

A. 6 Machines
Total Running Time =
B. 11 Machines
Total Running Time =

Partitioning the data by uniqueCarrier appears to have worked well for both approaches. In the syslogs the Reduce Input Group is set to 20. Indicating one data partition per uniqueCarrier. This is a good design choice that has lead to proper load balancing.

In terms of performance and scalability, I assume that HCOMPUTE is better. The partitioning of the data into regions and easy division in range partitioning by date makes the approach more scalable and more parallelizable ultimately leading to better performance. The HBase approach retains its optimized structure while being highly flexible. For instance, if we suddenly wanted average flight delays for the year 2009, the current implementation would require minimal change.

In terms of coding, the two approaches were comparable. Once the data is obtained from HBase, HCOMPUTE and SECONDARY are nearly identical in terms of map/reduce design, so the effort was the same. Getting the data into HBase by writing HPOPULATE was not too tricky. The main hurdle was thinking about table creation in combination with a mapper.

The setup effort was much, much worse for the HBase approach. Setting up HBase locally was not easy. I spent a day figuring out that I needed to specify a clientPort AND quorum in

hbase-site.xml for it to work. The AWS analysis has been a nightmare. All of my runs last indefinitely with the following stderr message :

```
AttemptID:attempt_1521337626773_0001_m_000008_0 Timed out after 600 secs
Container killed by the ApplicationMaster.
Sent signal OUTPUT_THREAD_DUMP (SIGQUIT) to pid 9020 as user hadoop for
container container_1521337626773_0001_01_000010, result=success
Container killed on request. Exit code is 143
Container exited with a non-zero exit code 143
```

I read of a similar error message on StackOverflow (https://stackoverflow.com/questions/30533501/hadoop-mapper-is-failing-because-of-container-killed-by-the-applicationmaster) indicating that my application exceeded the memory capacity. However, that seems unlikely. I've selected m3.xlarge (vCore = 8,Memory = 15 GB, storage = 80 GiB) for my master and core nodes. I even selected m4.xlarge for master and core in the hopes that it would solve my problem. However, I get the same timeout result. With memory issues still in mind, I considered that inserting all of the attributes for a flight might be causing the memory trouble, I tried only inserting a couple of attributes (date and arrivDelayMinutes) for test purposes. But, that didn't work : I got the same timeout result.

In addition, it's occured to me that perhaps somehow my HBase configuration is incorrect. My IDE indicates that quite a few of the classes I'm using are deprecated. However, I've found numberous solutions online that use a similar setup with success. I've included two approaches that I came up with in the HPOPULATE directory.