**Name:** Khoa Cu Dang (Cody) Cao

**Course:** CS6240 Spring 2024

Homework 4 Report

# Source Code

### Utility Classes

These classes are used commonly between the HBase-Compute and Secondary Sort program.

**Common**

This class defines the constants used by both program, as well as a function to validate the flight records.

**A screen shot of a computer

Description automatically generated**

**FlightHeader**

This class defines the columns being used from the data

**A screenshot of a computer program

Description automatically generated**

**FlightGroup Comparator**

This class is used as a grouping comparator class in order to group key-value pairs with similar carrier into the same reducer.



**FlightKey Comparator**

This class is used as a sort comparator class how output key-value pairs from the map are sorted.

A computer screen with text

Description automatically generated

**FlightPartitioner**

A screen shot of a computer program

Description automatically generated

**FlightKey**

This class acts as the key for the map phase. It contains the String that contains the name of the airline, as well as the month that the flight record happened. It also contains two compare methods: one for comparing the airline name, the other comparing both the name and the month.

## A screenshot of a computer program Description automatically generated

## Reducer

Like other Utility classes, I’m using the same Reducer approach to Secondary Sort and H-Computer since it would make things easier for development, given the fact that the expected output from the map phase of two programs should be identical.

A screenshot of a computer program

Description automatically generated

## Mapper

**Secondary Sort**

**A computer screen shot of a program

Description automatically generated**

The map phase of Secondary Sort validates the data (including filtering records of flights happened in 2008) and emits the key, as well as the arrival delay minutes value.

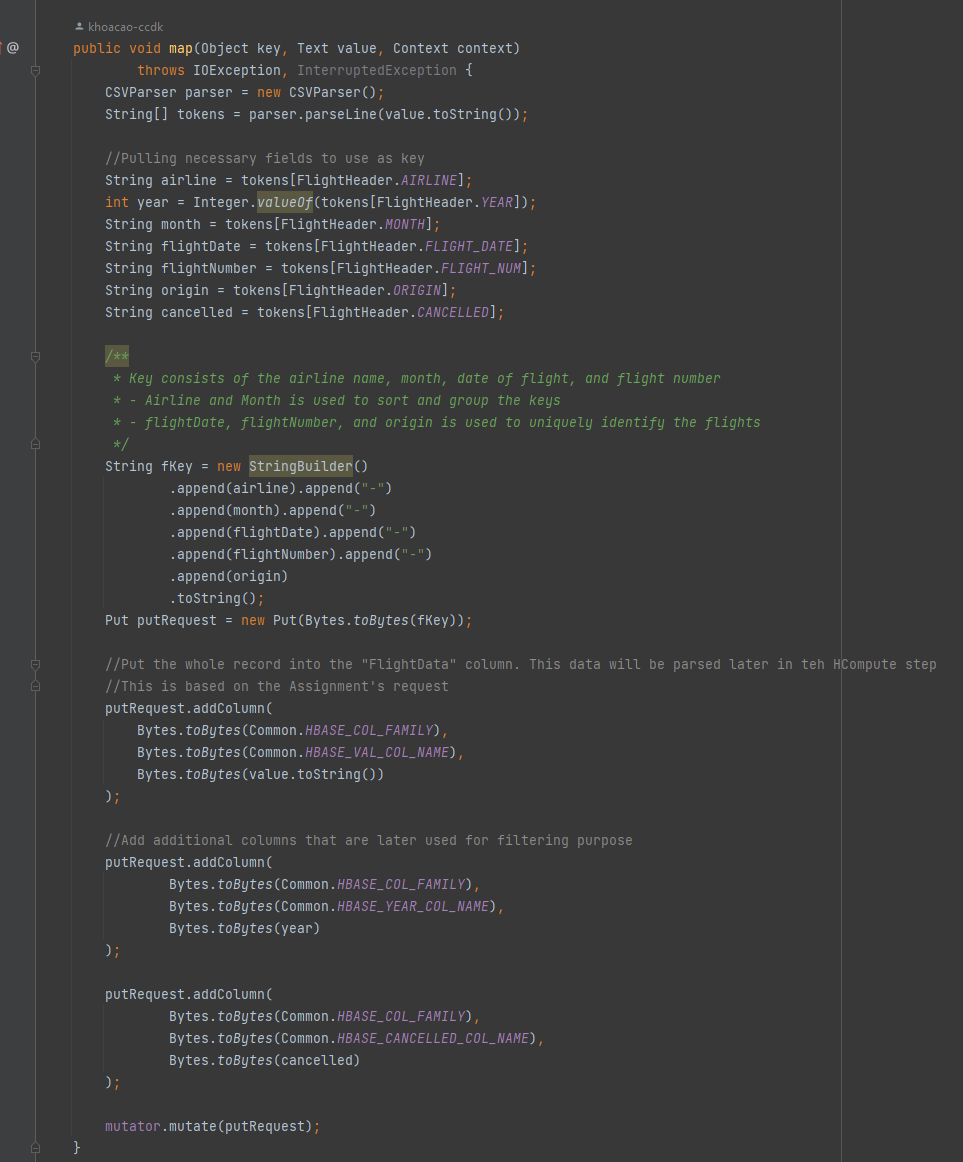
**H-Populate**

For each map instance, I set up a connection towards the table in HBase, as well as set up a one-minute flush period, which means the mappers would only write to HBase every minute.

**A computer screen shot of text

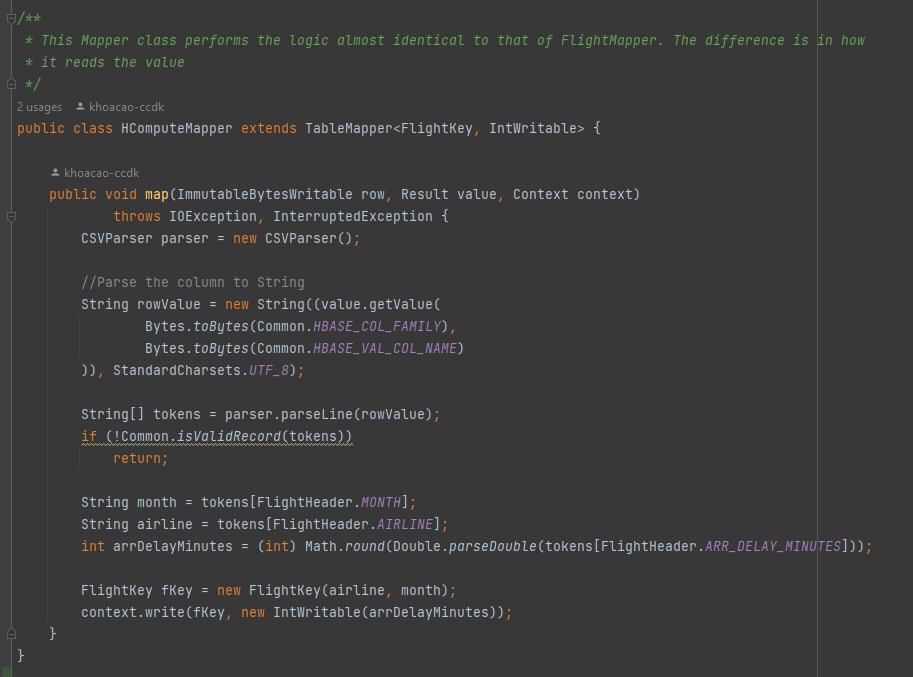
Description automatically generated**

For quick filtering in later stage, the map function would create some additional columns to the data row (year and cancelled), while also writing the whole record as an additional column according to the requirements of the assignment. No filtering efforts is performed at this stage.

****

**H-Compute**

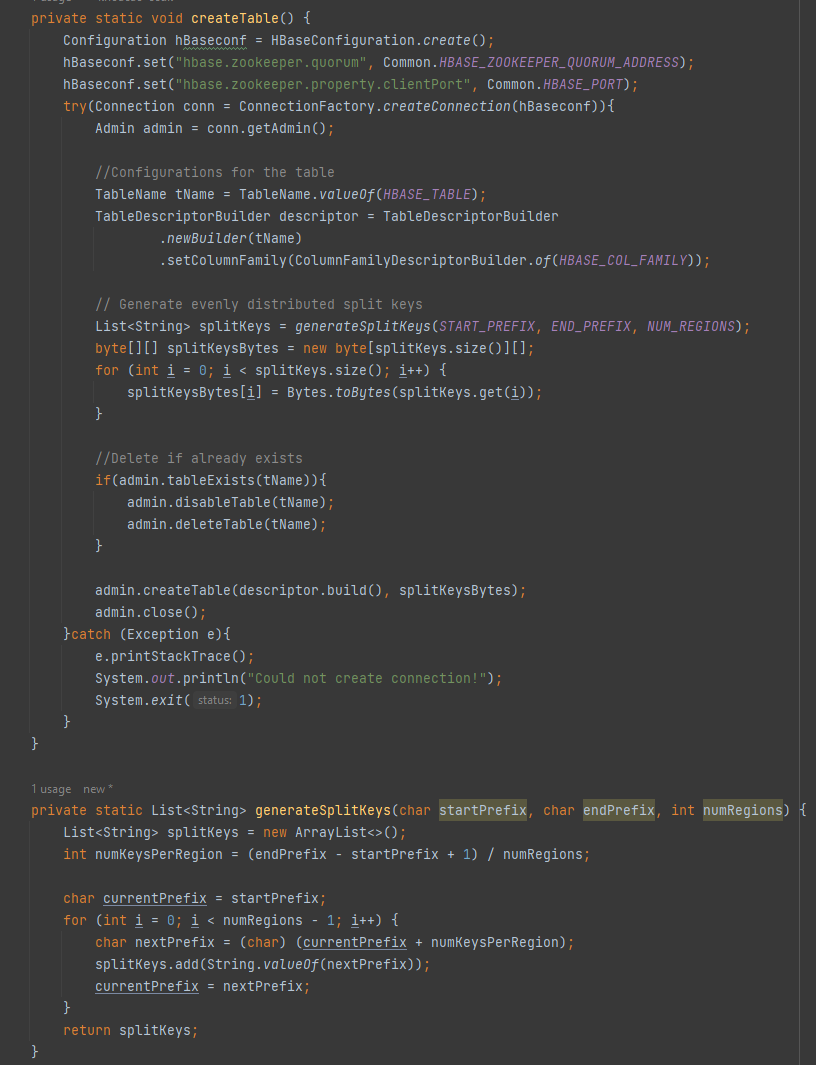
This essentially performs the same task as that of the SecondarySort program.

****

### Main program

**HPopulate**

Omitting the unnecessary parts, here is the logic to create the table in HBase.

****

**HCompute**

Here is the logic of setting up the mappers, as well as filtering unnecessary records.

****

# Pseudo code

### Secondary Sort

#### Mapper

For each flight record:

1. Check if the record is valid (no necessary field missing, the year is 2008)
2. Construct a key (FlightKey) from the airline name and the month value.
3. Emit the key and the arrDelayMinutes value.

#### Grouping Comparator

Group keys in accordance with the name of the airline. Thus, keys with the same airline name are grouped together and sent towards the same reducer.

#### Sort Comparator

Sort key in accordance with both name of the airline and the month value. This allows grouped key to further sorted and grouped into individual months within 2008.

#### Reducer

For each key-value pair:

1. Check if the month value of the key is the same as the currently considered month.
   1. If true:
      1. totalDelayMinutes += delay minutes
      2. numRecordCounted++
   2. If false:
      1. Calculate the average delay minute value of the previous month.
      2. Set the value currently considered month to the key’s month value.
2. For each month within the year, format their String (month: average delay minutes) and append it to the output String.
3. Emit the result.

### HPopulate

#### Mapper

1. At the setup step, create a connection towards HBase. Setup flush period to 1 minute.
2. For each flight record:
   1. Extracts airline, month, flight date, flight number, and origin for key to uniquely identify a record.
   2. Add the whole record line as a column.
   3. Add two additional columns (year and cancelled) for filtering later.
   4. Add to the buffered write queue.
3. At the cleanup step, flush any pending writes, then close the connection.

### HCompute

1. Setup a connection towards HBase.
2. Setup filter rule (year = 2008, not cancelled).
3. Setup scan and add filter rules.
4. Initiate mapper jobs for the scan.

#### Mapper

For each record given:

1. Validates the records (not missing necessary values)
2. Construct a key (FlightKey) from the airline name and the month value.
3. Emit the key and the arrDelayMinutes value.

#### Grouping Comparator

Group keys in accordance with the name of the airline. Thus, keys with the same airline name are grouped together and sent towards the same reducer.

#### Sort Comparator

Sort key in accordance with both name of the airline and the month value. This allows grouped key to further sorted and grouped into individual months within 2008.

#### Reducer

For each key-value pair:

1. Check if the month value of the key is the same as the currently considered month.
   1. If true:
      1. totalDelayMinutes += delay minutes
      2. numRecordCounted++
   2. If false:
      1. Calculate the average delay minute value of the previous month.
      2. Set the value currently considered month to the key’s month value.
2. For each month within the year, format their String (month: average delay minutes) and append it to the output String.
3. Emit the result.

# Proof of EMR execution

A screenshot of a computer

Description automatically generated



#### Added HBase input splits

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Performance Comparison

Here I’m comparing the results between using HBase vs MapReduce HDFS.

A screenshot of a computer

Description automatically generated

### HPopulate

An interesting thing that I found when running HPopulate is that it takes quite a long time to populate the data, despite me using buffered write and only opening/closing connection towards HBase once for every map instance. When testing with either 6 or 11 instances on EMR (m1.large), I find that it hovers around 15 minutes to read data from S3 and write to HBase region server. The HBase table size is also larger than the original data in HDFS (5.03Gb vs 4Gb). This can be explained by me creating redundant columns, as well as a complex key structure to uniquely differentiate flight records).

What I’m doing is to pre-define table splits based on the alphabet. The reason is based on the way I’m constructing the keys (airline-month-date-flightnumber-origin). Airlines are a string with two characters. Thus, I believe that separating them this way would yield a better performance. I tried running the program with and without the pre-defined split conditions and it yields a better performance (15 minutes with vs 20 minutes without). Interestingly, the result stays the same regardless of the cluster size.

Additionally, when running 6 instances vs 11 instances, there is a reduction in the runtime of HPopulate (20 minutes vs 15 minutes)

### HCompute vs Secondary Sort

When strictly comparing reading from HBase vs HDFS, seems like the HBase scan with filter functionality performs better, given that I created redundant columns for filtering effort, which reduces the number of data being transferred between the region server and the map instance, thus reduce the number of input records in the map task (13,395,076 for Secondary Sort vs 5,824,423 for HCompute).

However, considering the additional effort of setting up HPopulate and the time it takes to write data to HBase, I’d still consider the Secondary Sort approach, unless there is a use case where after the data is put into HBase, I can spin up multiple programs to read and create different outputs.

Generally, the time trade-off between the two approaches is as follow:

* **Secondary sort:** # of compute tasks (**α**)\* runtime per task (a)
* **HBase:** HPopulate runtime (b) + # of compute tasks(**α**) \* runtime per compute task (c)

If there is large enough number of compute task (**α**) to makes the HBase method faster, HBase might worth the extra development effort.