CSE 414 Homework 7: Parallel Data Processing and Spark

Objectives: To write distributed queries. To learn about Spark and running distributed data processing in the cloud using AWS.

Assigned date: Saturday, Dec 5, 2020.

Due date: Sunday, Dec 13, 2020, at 11 pm.

What to turn in:

Your Spark code in the SparkApp.java file with createLocalSession() enabled.

Resources:

- Spark programming guide
- Spark Javadoc
- Amazon web services EMR (Elastic MapReduce) documentation
- Amazon S3 documentation
- **Starter code files**

Spark Programming Assignment (75 points)

In this homework, you will be writing Spark and Spark SQL code, to be executed both locally on your machine and also using Amazon Web Services.

We will be using a similar flight dataset used in previous homework. This time, however, we will be using the *entire* data dump from the US Bureau of Transportation Statistics, which consists of information about all domestic US flights from 1987 to 2011 or so. The data is in Parquet format. Your local runs/tests will use a subset of the data (in the flights_small directory) and your cloud jobs will use the full data (stored on Amazon S3).

Your main task is to complete the Spark programming locally. Once your code works locally, you can follow our instructions to get it running on AWS, but this is optional.

Here is a rough outline on how to do the HW:

A. **(Optional)** Sign up for AWS and apply for credit for AWS

B. Complete the HW locally

C. **(Optional)** Run your solutions on AWS Elastic MapReduce one at a time when you are fairly confident with your solutions

A. (Optional) Sign up on Amazon Web Services

Follow these steps to set up your Amazon Web Services account.

1. If you do not already have an Amazon Web Services account, go to AWS Educate and sign up with your @uw email.
2. Click "Join AWS Educate", and choose the student account. Then complete the necessary forms, and wait until your account is approved. **This may take a while, so do this early.**
3. To get $$$ to use Amazon AWS, sign in to your AWS Educate account, click "AWS Account", and create a starter account. **This will only work if you have not applied for a starter account in the past.**
4. Follow the steps to finalize your credits. If everything works, you should have $100 credit in your account.

IMPORTANT: If you run AWS in any other way rather than how we instruct you to do so below, you must remember to manually terminate the AWS clusters when you are done. While the credit that you receive should be more than enough for this homework assignment, you will be responsible for paying the extra bill should your credits be exhausted.

Now you are ready to run applications using the Amazon cloud. But before you do that let's write some code and run it locally.

B. Get Code Working Locally

We have created empty method bodies for each of the questions below (QA, QB, and QC). *Do not change any of the method signatures*. You are free to define extra methods and classes if you need to. We have also provided a warmup method that shows fully-functional examples of three ways that the same query could be solved using Spark's different APIs.

There are many ways to write the code for this assignment. Here are some documentation links that we think would get you started up about what is available in the Spark functional APIs:

- Spark 2.4.5 Manual
- Spark 2.4.5 Javadocs
- Dataset
- Row (see also RowFactory)
- JavaRDD (see also JavaPairRDD)

- Tuple2

The quickstart documentation also has more depth and examples of using RDDs and Datasets.

For questions a, b, and c, you will get the points for writing a correct query.

(a) (15 points) Complete the method QA in SparkApp.java. Use the Spark functional APIs or SparkSQL. Select all flights that leave from 'Seattle, WA', and return the destination city names. Only return each destination city name once. Return the results in an RDD where the Row is a single column for the destination city name.

(b) (30 points) Complete the method QB in SparkApp.java. Only use the Spark functional APIs. Find the number of non-canceled (!= 1) flights per month-origin city pair. Return the results in an RDD where the row has three columns that are the origin city name, month, and count, in that order.

(c) (30 points) Complete the method QC in SparkApp.java. Only use the Spark functional APIs. Compute the average delay from all departing flights for each city. Flights with NULL delay values should not be counted, and canceled flights should not be counted. Return the results in an RDD where the row has two columns that are the origin city name and average, in that order.

Testing Locally

We provide cardinality testing when you run

```
$ mvn test
```

You are responsible for verifying you have the correct format and contents in your results.

Running Local Jobs

To actually execute the main method, toggle the SparkSession initialization on lines 147 and 148 of SparkApp.java to allow it to run locally (local SparkSession, not cluster). Run from the top-level directory (with pom.xml in it):

```
$ mvn clean compile assembly:single
$ java -jar target/sparkapp-1.0-jar-with-dependencies.jar \
      flights_small output
```

Note that, on Windows, this should be executed in the root directory of this repo, so that the program can find bin/winutils.exe. (The code uses the directory of execution .; you can change

this in the createLocalSession()method if you must.) For reference, winutils.exe was obtained for this Hadoop version from a Github repo.

For this quarter, we added code to add compatibility with Java 9+. It should work fine. In case there is a problem, you can force a Java 8 execution by downloading a Java 8 JRE and setting your JAVA_HOME variable to your Java 8 runtime.

C. (Optional) Run Code on Elastic Map Reduce (EMR)

Run your jobs on Elastic Map Reduce (EMR) as described below, and copy the resulting output from EMR to QA.txt, QB.txt, and QC.txt, respectively.

Running all jobs at the same time with the provided configuration took less than 30 min for the solutions.

We will use Amazon's Elastic Map Reduce (EMR) to deploy our code on AWS. Follow these steps to do so after you have set up your account, received credits as mentioned above, and have tested your solution locally. **Read this carefully!**

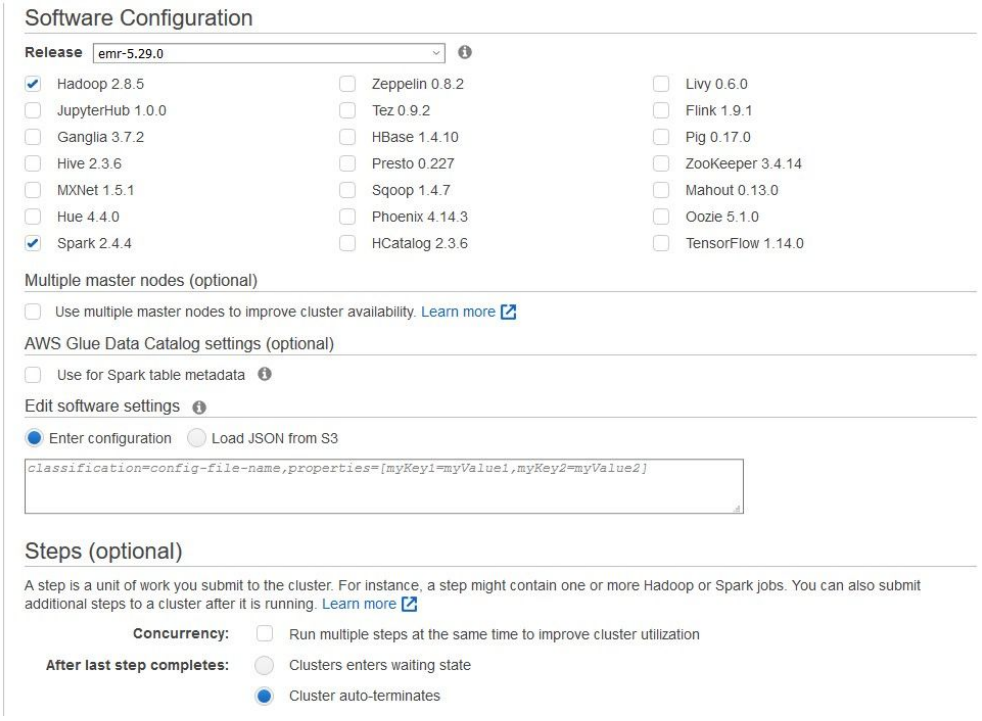
1. Toggle the SparkSession initialization of SparkApp.java to allow it to run on AWS (cluster SparkSession; comment out local). Then create a jar file from the top level directory that packages everything needed to run the Spark application. The following command creates the jar file in the targets folder:

   ```
   $ mvn clean compile assembly:single
   ```

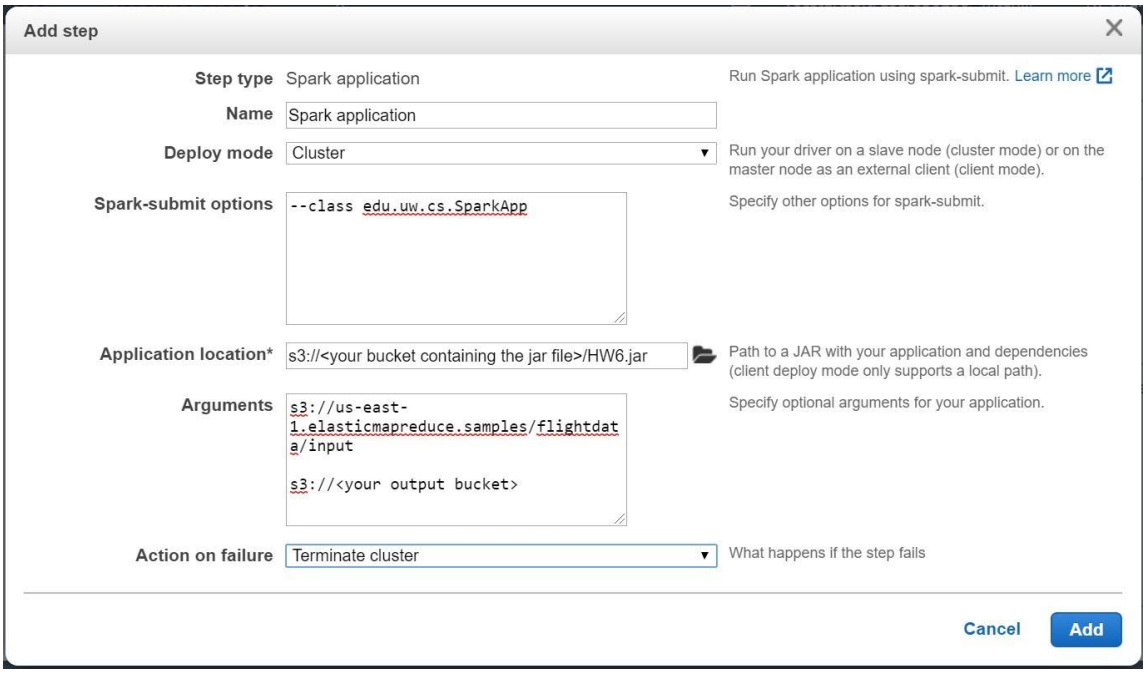
2. Login to S3 and create a "bucket". S3 is Amazon's cloud storage service, and a bucket is similar to a folder. Give your bucket a meaningful name and select **US East (N. Virginia)**, and leave the other settings as default. Upload the jar file that you created in Step 1 to that bucket by selecting that file from your local drive and click "Upload" once you have selected the file.

3. Login to EMR. Make sure you select US East (N. Virginia) on the upper right. **This is the only region supported by the starter account.**

4. We will first configure cluster software. Click on the “Create Cluster" link, then click "Go to advanced options” in the Amazon EMR console.
   - **Check the boxes for Spark and Hadoop** so that your screen looks like this:



- Next, scroll to the Steps section at the bottom of the page and **create a Spark application step**. A "step" is a single job to be executed. You could specify multiple Spark jobs to be executed one after another in a cluster. Fill out the Spark application step details by filling in the boxes so that your screen looks like this (with SparkApp instead of HW6):



The --class option under "Spark-submit options" tells Spark where your main method lives. (--class edu.uw.cs.SparkApp)

The "Application location" should just point to where your uploaded jar file is. You can use the folder button to navigate.

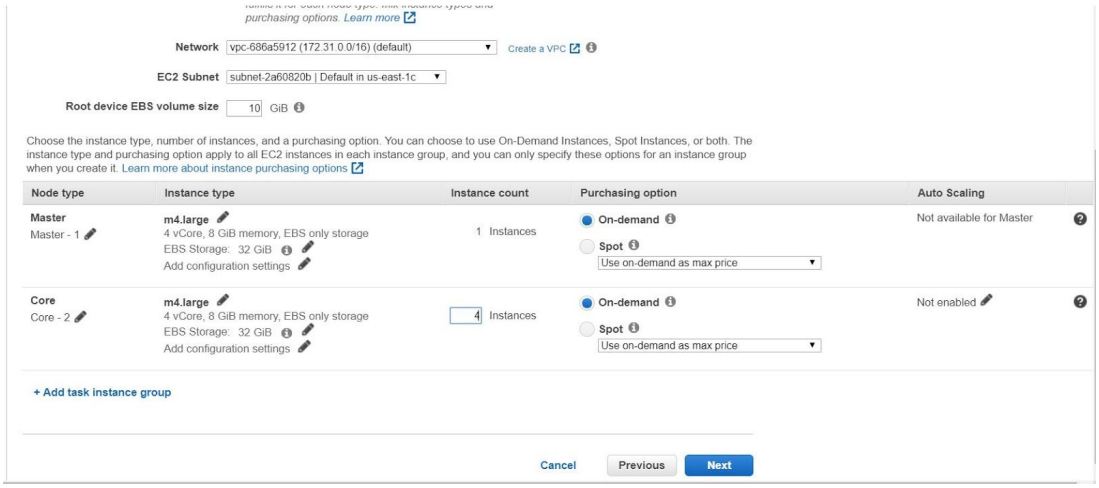
The full flights data location is the first argument: s3://us-east-1.elasticmapreduce.samples/flightdata/input

The output destination is the second argument. Use can use the bucket that holds your jar. You can modify the “output” folder name prefix to be something different if you like.

Make sure you fill out the correct bucket names. There are two arguments listed (and separated by white space, as if you were running the program locally):

Change “Action on failure” to “Terminate cluster” (or else you will need to terminate the cluster manually).

- **Click Add.**
- Back to the main screen, now on the After last step completes: option at the bottom of the page, select Cluster auto-terminates so the cluster will shut down once your Spark application is finished. Click Next.
- On the next screen, we will now configure the hardware for a five-node EMR cluster to execute the code. We recommend using the “m4.large” "instance type", which is analogous to some set of allocated resources on a server (in AWS terminology, “m” stands for high memory, “4” represents the generation of servers, and “large” is the relative size of allocated resources). You get to choose how many machines you want in your cluster. For this assignment 1 master instance and 4 core (i.e., worker) instances of m4.large should be good. You are free to add more or pick other types, but make sure you think about the price tag first... Grabbing 100 machines at once will probably drain your credit in a snap :(If m4.large is not available, choose another instance with a similar name (m4.xlarge, m5.large, etc.). **Click Next.**



- Under “General Options” *uncheck the “Termination protection” option*. We recommend that you allow the default logging information in case you need to debug a failure. **Click Next.**
- Click *Create cluster* once you are done and your cluster will start spinning up!

It will take a bit for AWS to both provision the machines and run your Spark job. As a reference, it took about 10 mins to run the warmup job on EMR. You can monitor the status of your cluster on the EMR homepage.

To rerun a similar job (maybe you want to try a different jar), use the "Clone" cluster button to copy the settings into a new job when you run your actual HW problems.

Make sure your cluster is terminated! It should do so if you selected the options above. You should check this each time you look at the HW, just to make sure you don't get charged for leaving a cluster running. It's fine if you see warning (or even occasional error) messages in the logs. If your EMR job finishes successfully, you should see something similar to the below in the main EMR console screen:

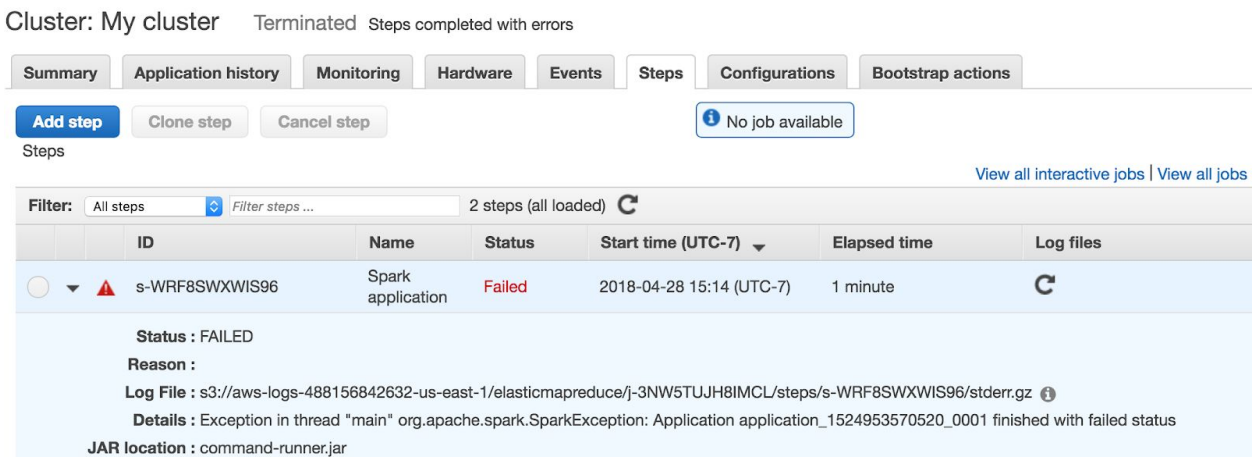
j-2BIXJO3MBV2B7	Terminated All steps completed	2017-11-08 17:51 (UTC-8)	17 minutes	24

Debugging AWS jobs

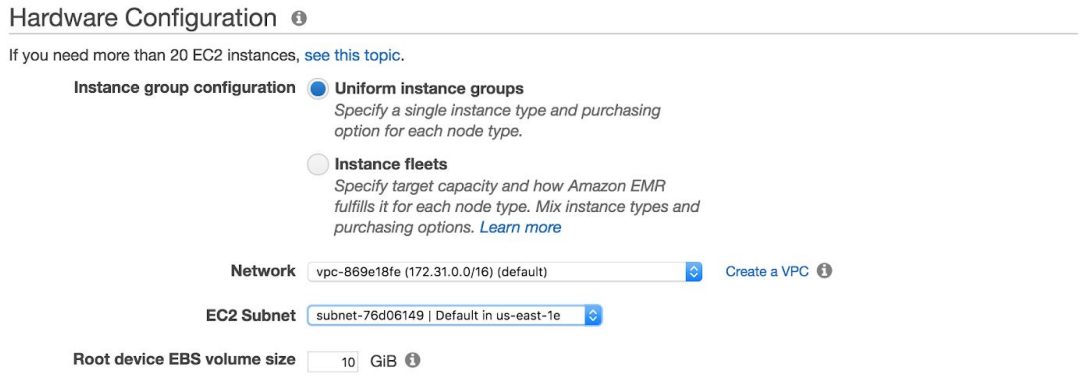
Debugging AWS jobs is not easy for beginners. Besides making sure your program works locally before running on AWS, here are some general tips:

- Make sure that you set ALL the job details (i.e., options, arguments, bucket names, etc) correctly!
- Make sure you switched the two lines of SparkSession code mentioned to run your job on AWS instead of locally.

- Make sure you freshly compile your solution and replace your jar to test a new version of code!
- **99% of cluster failures or errors are due to the first three points!**
- The easiest way to debug is to look at the output/logging files. Spark generates a lot of log files, the most useful ones are probably the stderr.gz files listed under containers/application.../container/stderr.gz. You will have one container folder per machine instance. So make sure you check all folders. You should also check out the log files that you have specified when you created the job in Step 8 above. You can also see the names of those files listed as "Log File" under "Steps":



- It is rare that your HW solution is fine but the cluster fails. This is usually due to AWS not being able to grab your machines due to the demand for the instance type saturating the supply available. If you can't find available instances in a region, try changing to a different **EC2 subnet**, like so:



- Spark has a web UI that you can set up to check on job progress etc. You can check out their webpage for details. But these are more involved so you are probably better off to first try examining the logs. Specifically, try the "Application History" tab and the dropdown.

IMPORTANT: Cleanup after completing the HW

Double-check that the clusters you have created are all terminated.

S3 charges by downloading/uploading data from/to the buckets. So once you are done with the assignment you might want to delete all the buckets that you have created (in addition to shutting down any EMR clusters that you have created).

The amount you are allocated from Amazon should be more than enough to complete the assignment. And every year we have students forgetting to shut down their cluster/clean up their buckets and that can result in substantial charges that they need to pay out of pocket. So be warned!

Submission Instructions

Turn in your SparkApp.java by submitting it to Gradescope.

Note: if you changed your code to run on AWS with the createClusterSession() enabled, make sure to change it back to createLocalSession() before submitting