Homework 6 | Database Internals

CSE 344 - Introduction to Data Management

Due date: Sunday, August 18th at 11pm

After pulling from upstream, you will find the starter files in the “hw6” folder of your hw6 repo. While we provide a testing framework for your queries, the testing we provide ONLY does cardinality checking. It is up to you to print/debug your results to check if it makes sense.

For the written portions, save your solutions in a PDF and name it “hw6.pdf” in the root directory of your GitLab repo.

1. Transactions - Precedence Graph (3 points)

Is the following schedule conflict serializable? Draw a precedence graph for the schedule and, if the transaction is serializable, write down an equivalent serial order of the transactions.

$R_1(A)$, $R_2(A)$, $W_1(B)$, $W_2(A)$, $W_3(A)$, $R_1(C)$, $W_2(C)$, $R_2(C)$, $R_3(A)$, $W_3(A)$

2. Transactions - Schedule (7 points)

Imagine we are using strict two-phase locking (strict 2PL) Write a possible schedule for the below transactions given the initial locks provided in the table. Indicate when any new locks are acquired (ex: L(A)) or released (ex: U(A)). (We are only using exclusive locks, not shared locks.) If a deadlock occurs, indicate when the transaction is aborted and retried. Write the schedule in table form (not inline).

T1: W(B), R(B), W(A), W(A) COMMIT
T2: R(A), W(A), COMMIT

T1	**T2**
L(B)	
W(B)	
	L(A)
	R(A)
...	...

(you may use more more rows as needed)

3. Spark and EMR (30 points)

AWS Account Setup

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

1. If you do not already have an Amazon account, go to their website and sign up. Note: Amazon will ask you for your credit card information during the setup process. **If you follow these instructions and do local testing first you should NOT need to spend any money.** Amazon may perform a credit card check by charging and then refunding one dollar.
2. Sign in to your AWS console, go to "Support -> Support center" in the navigation bar, and locate your account number.
3. To get $$$ to use Amazon AWS, you must apply for credits by going to their education website. **You must use your UW email address, <your_uwid>@uw.edu, when registering for the credits, as they use this to verify your identity.** Leave the promo code blank, and enter your AWS account number on the next page. Make sure you do NOT check the starter account option on the final page as that has limited permissions which may cause problems.
4. After applying, you will have to wait to be approved. You should get an email when your application has been approved, which gives you a credit code. Make sure you double check your spam folder. Once you have it, go to AWS's credit management and enter the credit code. We have no control over how long this can take, but was told it can range from seconds to a few days. While waiting for your credit, you can write the code locally.

IMPORTANT: If you exceed the credit you are given, Amazon will charge your credit card without warning. If you run AWS in any other way rather than how we instruct you to do so below, **you must remember to terminate the AWS EMR clusters when you are done**. While the credit that you receive should be more than enough for this homework assignment, but you should still monitor your billing usage by going to their billing website and clicking on "Bills" (upper left). You should get $100 from AWS once your application is approved. The credits that you have left over after the assignment are for you to keep, but **if you exceeded the credits due to forgetting to turn off your clusters, mining Bitcoin, etc. then you will be responsible for paying the extra bill.**

You are now ready to run applications using AWS. But before you do that let's write some code and run it locally.

Problems

We have created empty method bodies for each of the questions below (Q3A, Q3B, and Q3C). **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 fully implements three ways that the same query could be solved in practice.

Save the resulting output from EMR to Q3A.txt, Q3B.txt, and Q3C.txt, respectively. Running all jobs at the same time with the provided configuration took less than 30 min for the solutions.

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 2.4.2 Javadocs
- Dataset
- Row (see also RowFactory)
- JavaRDD (see also JavaPairRDD)
- Tuple2

For parts a, b, and c, you will get the points for writing a correct query. For part d you will get the full points only if both your queries are correct and you have the correct output from running the full dataset on AWS in your Q3A.txt, Q3B.txt, and Q3C.txt files.

(a) (6 points) Complete the method Q3A in HW3.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) (10 points) Complete the method Q3B in HW3.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) (10 points) Complete the method Q3C in HW3.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.

(d) (4 points)
Run your jobs on Elastic Map Reduce (EMR) as described below, and copy the resulting output from EMR to Q3A.txt, Q3B.txt, and Q3C.txt, respectively.

Running Local Jobs

We provide cardinality testing when you run

```
$ mvn test
```

To actually execute the main method, **toggle the SparkSession initialization on lines 147 and 148 of HW3.java to allow it to run on locally** (local SparkSession, not cluster). Run from the hw6 folder:

```
$ mvn clean compile assembly:single
$ java -jar target/hw6-1.0-jar-with-dependencies.jar \
      flights_small output
```

You also need to run mvn and the java runtime with Java 8 and not a later version of Java. To force this to be the case, preface your “mvn … “ command with a command to set your JAVA_HOME environment variable to point to your Java 8 runtime. So for example on Mac OS X with Java 1.8.0, you would run:

```
$
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_131.jdk/Contents/
Home/ mvn clean compile assembly:single
```

Running EMR Jobs

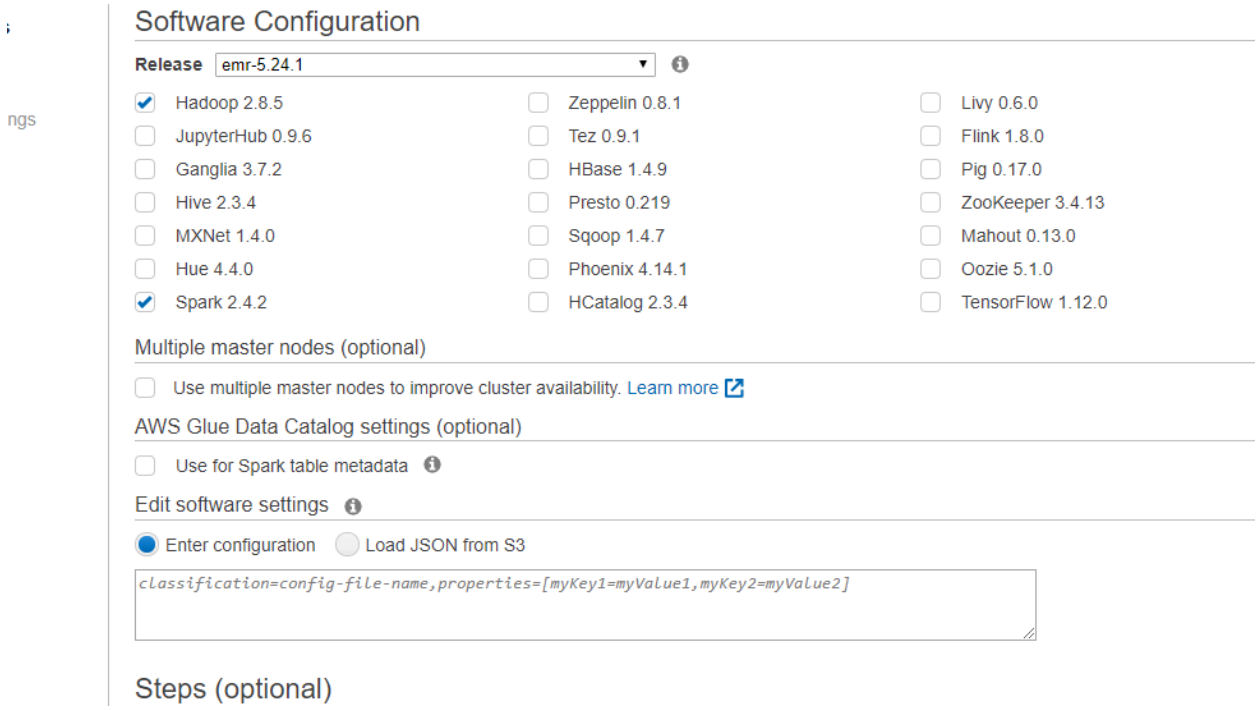
We will use Amazon's Elastic Map Reduce (EMR) to deploy our code on AWS. Follow these steps to do so after you have set up your account, received credits as mentioned above, and have tested your solution locally. **Read this carefully!**

1. **Toggle the SparkSession initialization on lines 147 and 148 of HW6.java to allow it to run on AWS** (cluster SparkSession, not local, make sure to comment out local). Then create a jar file from the hw3 directory that packages everything needed to run the Spark application. The following command create the jar file in the targets folder:

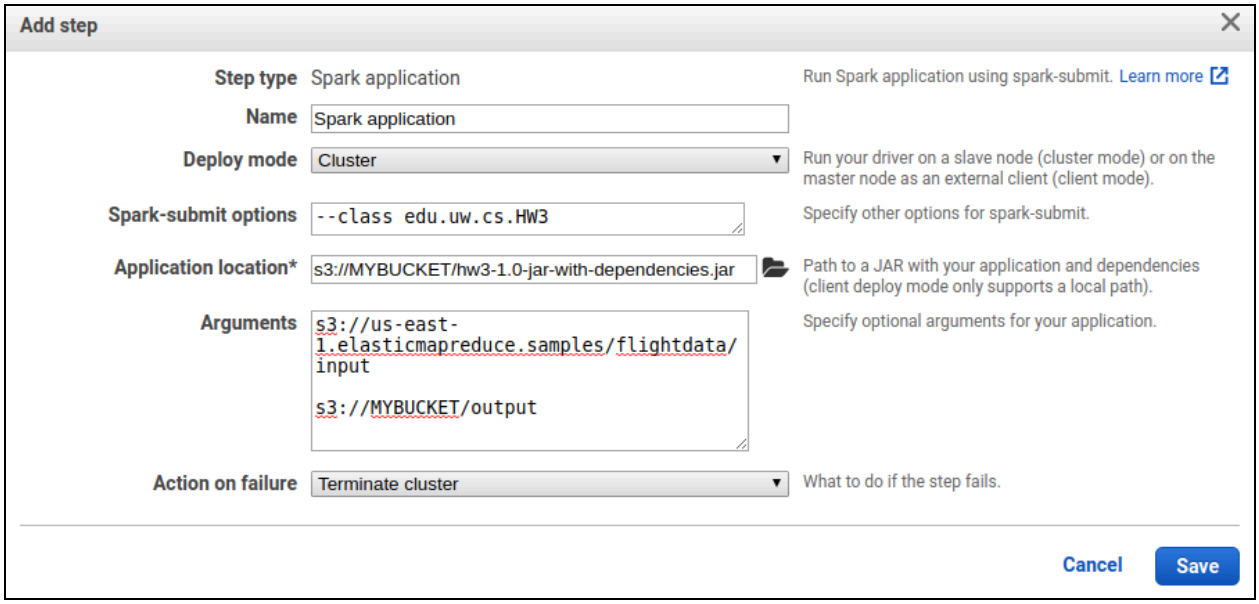
   ```
   $ mvn clean compile assembly:single
   ```

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 leave the 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) or US East (Ohio) on the upper right.** This is because the full, public data file that we will use is stored there, so it will be faster to access from a machine located nearby.
4. We will first configure cluster software. **Click on the "Create Cluster – Advanced Options"** link in the Amazon EMR console. **Check the boxes for Spark and Hadoop** so that your screen looks like this:



5. 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 can 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 HW6 instead of HW3):



The --class option under "Spark-submit options" tells Spark where your main method lives.

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 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):

6. **Change “Action on failure” to “Terminate cluster”** (or else you will need to terminate the cluster manually). **Click Add**.
7. Back to the main screen, now **check the “Auto-terminate cluster after the last step is completed” option** at the bottom of the page, so the cluster will automatically shut down once your Spark application is finished. **Click Next**.
8. 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**.

9. 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**.
10. 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 you terminate the cluster! 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:

If everything worked out, congrats! You just successfully rented time to run an application on a network of computers across the country!

Debugging EMR Jobs

Debugging jobs on the cloud 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 the ALL job details correctly (options, arguments, bucket names, etc.). Take another pass or two at the spec if you haven’t already.
- Make sure you switched the two lines of code mentioned to run your job on AWS instead of locally.
- Make sure you freshly compile your solutions and replace your jar to test a new version of your Spark application!
- **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 and find a stack trace of your error. Spark generates a lot of log files, the most useful ones are probably the stderr.gz file. You

will find it if you click into the cluster details, click into the “Steps” tab, and look at the “Log Files” for any of your steps.

It is rare that your HW solution is fine but the cluster fails. This is usually due to AWS not being able to provision your machines due to abnormally high demand. Sidestep this problem by waiting for a bit or by using another data center (in AWS terms, Availability Zone or AZ) by selecting from the top right corner of the AWS console.