CS44800 - Project 5
Big Data Management Using Hadoop/Spark
Spring 2023

Due: Sunday, April 23, 2023, 11:59PM
Total Points: **5 points**

Learning Objectives

1. Setup Hadoop Distributed Filesystem (HDFS) and Spark
2. Use HDFS to store input and output data files
3. Use Spark to perform basic query processing over data files stored in HDFS

Project Descriptopn

In this project, you will setup and use Hadoop and Spark to perform data processing tasks. You are going to setup HDFS and use it for data storage. You will implement a given set of queries using Spark as your data processing framework. This project will give you experience loading data to HDFS and writing and executing Spark applications. To familiarize yourself with Hadoop and Spark, we recommend watching these short videos: Hadoop Video and Spark Video before you start.

Scholar Cluster

The scholar cluster provides an environment to perform Big Data processing using Hadoop and Spark frameworks. Use the web-based Remote Desktop from any web browser using the following link:

`https://desktop.scholar.rcac.purdue.edu/`

Note: Every registered student in the class should have access to Scholar. If you have difficulties accessing Scholar, contact RCAC (rcac-help@purdue.edu) as soon as possible.

More information about Scholar can be found here:

`https://www.rcac.purdue.edu/compute/scholar`

Dataset

For this project, we will use the **MovieLens** dataset. The schema is given in Table 1. It will be helpful to familiarize yourself with the data encoding by reading the following README file:

`http://files.grouplens.org/datasets/movielens/ml-1m-README.txt`

File Name	Schema
users.dat	(UserID::Gender::Age::Occupation::Zipcode)
movies.dat	(MovieID::Title::Genres)
ratings.dat	(UserID::MovieID::Rating::Timestamp)

Table 1: Schema

Part 1: Setting up Hadoop and Spark on Scholar

The first part of the project involves setting up HDFS and Spark in Scholar. We have provided you with instructional videos to help you set this up.

1. Follow the video to setup passwordless SSH
   (a) Generate RSA keys using `ssh-keygen -t rsa`
   (b) Copy public key to authorized keys `cat .ssh/id_rsa.pub >> .ssh/authorized_keys`
2. Follow the video to setup HDFS
   (a) Download and unzip Hadoop 3.3.5 binary from https://hadoop.apache.org/releases.html
   (b) Configure `core-site.xml` in `hadoop-3.3.5/etc/hadoop`
   (c) Configure `hdfs-site.xml` in `hadoop-3.3.5/etc/hadoop`
   (d) Create `namenode` and `datanode` directories.
   (e) Configure `hadoop-env.sh` in `hadoop-3.3.5/etc/hadoop`
   (f) Format namenode using `./hdfs namenode -format` in `hadoop-3.3.5/bin/`
   (g) Start and Stop HDFS using `./star-dfs.sh` and `./stop-all.sh` in `hadoop-3.3.5/sbin/`
   (h) Check your Hadoop instance in the local browser at `localhost:9870`.
3. Follow the video to setup Spark
   (a) Download the Spark 3.3.2 binary from https://spark.apache.org/downloads.html
   (b) Configure `spark-env.sh` in `spark-3.3.2-bin-hadoop3/conf`
   (c) Configure `workers` in `spark-3.3.2-bin-hadoop3/conf`
   (d) Configure `log4j2.properties` in `spark-3.3.2-bin-hadoop3/conf`

NOTE: In case it is not already running, remember to start the HDFS using `start-dfs.sh` before running any `hdfs` or `spark` commands. Always stop the hdfs using `stop-all.sh` once you are done using it, or are about to log out of scholar.

Part 2: Loading data in to HDFS

Log in to the Scholar Cluster copy and unzip the data files into a temporary directory (e.g., ' `cs448p5` ') in your home directory.

```
mkdir cs448p5
cp /home/tbonjour/cs448/p5/p5-data.zip cs448p5/
cd cs448p5; unzip p5-data.zip
```

Once you have the unzipped files, follow the instructional video to load data into HDFS. Use the following export commands to avoid having to sepcify the full path to hdfs commands:

```
export PATH=<path to your hadoop binary>/sbin:$PATH
export PATH=<path to your hadoop binary>/bin:$PATH
```

Create a sub-directory to store the data files on HDFS using the following command (replace `<username>` with your username or Purdue career account):

```
hdfs dfs -mkdir -p /user/<username>/input
```

To upload data files into HDFS, use the following command:

```
hdfs dfs -put ./tmp/* /user/<username>/input
```

Now, you should have your data files in HDFS. Verify that by listing the files in the `input` directory, using the following command:

```
hdfs dfs -ls /user/<username>/input
```

The output of the above command should be similar to the following:

```
tbonjour@scholar-fe04:~/cs448/hadoop $ hdfs dfs -ls /user/tbonjour/input
Found 3 items
-rw-r--r--   1 tbonjour supergroup     171246 2023-04-11 19:17 /user/tbonjour/input/movies.dat
-rw-r--r--   1 tbonjour supergroup   24594131 2023-04-11 19:17 /user/tbonjour/input/ratings.dat
-rw-r--r--   1 tbonjour supergroup     134368 2023-04-11 19:17 /user/tbonjour/input/users.dat
tbonjour@scholar-fe04:~/cs448/hadoop $
```

Part 3: Warm-up Exercise

Follow the instructional video to run the warm-up exercise. In this exercise, you will perform the following tasks:

1. Download skeleton code from the course website.
2. Verify that the environment settings are valid.
3. Use Maven to prepare your Spark application to submit to the cluster.
4. Submit your application to the cluster using the example code in the skeleton Java project.
5. Save a screenshot of your output (with your **username** clearly visible) and include it with your submission as **warmup.png**.

The skeleton code is available from the course website. Download and extract the Maven-based Java project into a directory of your choice.

Maven Configuration

For this project, we use Maven to build Spark applications. Use the following command to activate Maven in your environment.

```
export PATH=/home/tbonjour/cs448/apache-maven-3.6.2/bin:$PATH
```

After that, use following command to check you're using the intended Maven setup.

```
mvn --version
```

Recall that Maven is a build tool which you use to compile your code. Re-running Maven to recompile your code is necessary to ensure that new changes in your code take effect after submitting the application to the Spark cluster.

Now, try to build the current skeleton code. The skeleton code comes with a feature to check if your development environment is configured correctly and it is called the Warm-up mode. First, build and package your Spark application using the following command:

```
mvn clean package
```

Note that clean deletes all files related to a previous build process invocation. The purpose of the above command is to package your Spark application into a JAR file (which is basically a container of all your Java code). The JAR file is located in the ./target directory of your maven-based project.

To be able to use the spark commands witout explicitly mentioning the whole path you can run the following command:

```
export PATH=<path to your spark binary>/bin:$PATH
```

This package is submitted to the Spark cluster using the following command:

```
spark-submit --class cs448.App target/cs448p5-1.0-SNAPSHOT.jar -i
"/user/<username>/input" -warmup
```

The `-i` argument specifies the input directory on HDFS. Note that `-warmup` argument causes the application to run the built-in warm-up exercise. Make sure to omit `-warmup` argument when running your code.

The warm-up will read each of the data files, count the number of lines, and print the number. If you have completed Part 1, 2 and 3 successfully, you should get the following output:

Take a screenshot of your output and save it as `warmup.png`. Make sure your username is clearly visible in the screenshot, similar to the figure above. You will include this with your submission.

Part 4: Spark Application (OPTIONAL)

Now, you are ready for your Spark application. **Your task is to implement a Spark application for the query given below.**

Query: Find all the distinct movie titles that are rated **greater than or equal to** `conf.q1Rating` and rated by users with occupation code **equal to** `conf.q1Occupation`.

/ example output **/**
`Problem Child (1990)`

Implementation

- Write all your code in `Porject5.java` given with the skeleton code.
- Implement the query in `runSparkApp1`.
- Save your output as text file in HDFS under the folder: `/user/<username>/output/query-1` where `<username>` is replaced with your username.
- Use Spark SQL to implement the queries. To save your result in HDFS, the *Dataset* with the results (from your SQL) has a method to store the data in HDFS. Another option is to transform the *Dataset* to a *RDD*, and from there save the data to HDFS. If you chose the latter, this should help.
- You may use `resolveUri` method from the `CS448Utils` class to build a full path for the input or the output file. For example, `CS448Utils.resolveUri(conf.inPath,conf.usersFName)` returns the full path to the user table input data file. See the warmup code example for more details.

Once you have implemented `runSparkApp1` function in the Project5 class, you can run the following command to execute the above query and store the output:

```
spark-submit --class cs448.App target/cs448p5-1.0-SNAPSHOT.jar -i
"/user/<username>/input" -o "/user/<username>/output" -q 1,12,3
```

Replace `<username>` with your username. Notice the additional parameters for specifying the query parameters. The argument `-q` has a comma-separated value which is parsed and passed to the Spark application to select which query is invoked and the respective arguments to use for the query. For example, `-q 1,12,3` indicates `runSparkApp1` is to be invoked with `occupation = 12`, and `rating >= 3`.

An instance of `App.Conf` class is passed to the method. This instance contains all the application configuration and the query parameters parsed from the command line arguments. You have the following public data members:

- `conf.inPath` : Path to the directory on HDFS containing data input files
- `conf.outPath` : Path to the directory on HDFS used to save data output files
- `conf.usersFName` : File name for the users table
- `conf.moviesFName` : File name for the movies table
- `conf.ratingsFName` : File name for the ratings table
- `conf.q1Occupation` : Integer value passed as the query parameter for occupation.
- `conf.q1Rating` : Integer value passed as the query parameter for rating.

Testing Your Code

The first step in order to test your code is to **load the expected output to HDFS**. You will load the data similar to what you did in part 2.

Log in to Scholar and copy the `testcases.zip` file in your home directory.

```
cp /home/tbonjour/cs448/p5/testcases.zip ~/
cd ~/; unzip ~/testcases.zip
```

Create a subdirectory to store the test case files on HDFS using the following command (replace `<username>` with your username or Purdue career account):

```
hdfs dfs -mkdir -p /user/<username>/test
```

To upload data files into HDFS, use the following command:

```
hdfs dfs -put ./testcases/* /user/<username>/test
```

Now, you should have your data files in HDFS. Verify that by listing the files in the `input` directory, using the following command:

```
hdfs dfs -ls /user/<username>/test
```

Next step is to **replace `<username>` with your username** in `TESTCASE_PATH` in `CS448Constants.java`

To test your distributed Spark application: simply append `-test` to your Spark submission command. For example, the following command will run your implemented application and test it against a predefined test-case:

```
spark-submit --class cs448.App target/cs448p5-1.0-SNAPSHOT.jar -i
"/user/<username>/input" -o "/user/<username>/output" -q 1,12,3 -test
```

NOTE: We may run your submission against other set of parameter values for `conf.q1Occupation` and `conf.q1Rating`. Make sure you are reading the parameters from `conf`, and not hard-coding any values.

What to Submit

1. **warmup.png** from **Part 3** with your username clearly visible in the screenshot of the output.
2. **OPTIONAL - Project5.java** file, with the implementation of the query from **Part 4**. If you need to create helper classes or any additional class, create those as inner classes inside this file. No additional file with any code should be submitted.

Copy both files in a single folder and zip it as `<username>.zip`. Replace `<username>` with your username.

Submit the zipped folder on Brightspace.

Rubric

Criteria	**Points**
Submitted **warmup.png**	1
Correct output and username visible in **warmup.png**	4
Correct implementation for **runSparkApp1**	0
Total Points	**5**

Table 2: Rubirc