**Spark and Hadoop**

In this assignment we will demonstrate our knowledge of Spark via the Spark Scala shell and the spark engine as observed using the Web UI.

You should turn in a PDF report with answers to the questions provided.

## Setup

You will some additional information to get started. Oh, here is a tip for using the show() method on a DataFrame. By default, the show command will truncate columns that are wide. Use the following to disable truncation.

df.show(false)

## Aggregation

You can use the DataFrame API to perform aggregation. I will give you an example here so you get the idea. You may want to consult with other references as well. Using the DataFrame API, we can perform aggregations on the results of a groupBy operation. One pattern is to alias the sql functions so you can reference them easily.

For the code snippet below, assume the DataFrame df is a listing of temperatures recorded on specific dates with columns low, high, and average. We want to get the lowest, highest and average temperature for each month.

import org.apache.spark.sql.{functions => f}

val newDf = df.groupBy($"month").agg(f.min($"low").alias("lowest"), f.max($"highest").alias("max"), f.avg($"average"))

### Joins

We can also perform joins using the DataFrame API. For cases where the column names are the same in both DataFrames, we can use the following syntax.

val joinedDf = df1.join(df2, ("colname"))

The resulting DataFrame will only have one column with the join key in it.

### Spark Shell

Start the Spark scala shell in local or cluster mode (your choice). In some cases, you may get better results running on Yarn.

## Movie Genre Analysis

We will do things a little differently for this assignment. We still want to join the movies with the ratings but we also want to break out the movies by genre and retain the rating statistics.

### Analysis Goal

We want to answer the following question:

Which movie has the most ratings in each genre? The output should have the following format:

genre, title, rating\_count, ave\_rating

### Approach

You are allowed to use either the DataFrame or SparkSQL API to perform operations. The order you do things may matter so you should experiment and evaluate results to make sure you are processing the data correctly.

You should consider creating a derived data set and then analyzing it to answer the question.

Q1. Describe your approach as operations performed on the data. Use terms that reflect the transformations you are making. Show the order you plan to do things.

1. Load the “movies.csv” and “ratings.csv” datasets into Spark DataFrames.
2. Join the exploded movies DataFrame with the ratings DataFrame on the movieId column.
3. Use the split() function to split the genres column in the movies DataFrame into an array of genres.
4. Use the explode() function to create a new DataFrame where each movie is repeated for each genre it belongs to.
5. Group the resulting DataFrame by genre and title, and compute the count and average of ratings for each group.
6. Join genre from maximum count of the rating scores to genres

### Programming

Work through the statements you need to have to prepare a dataset for query.

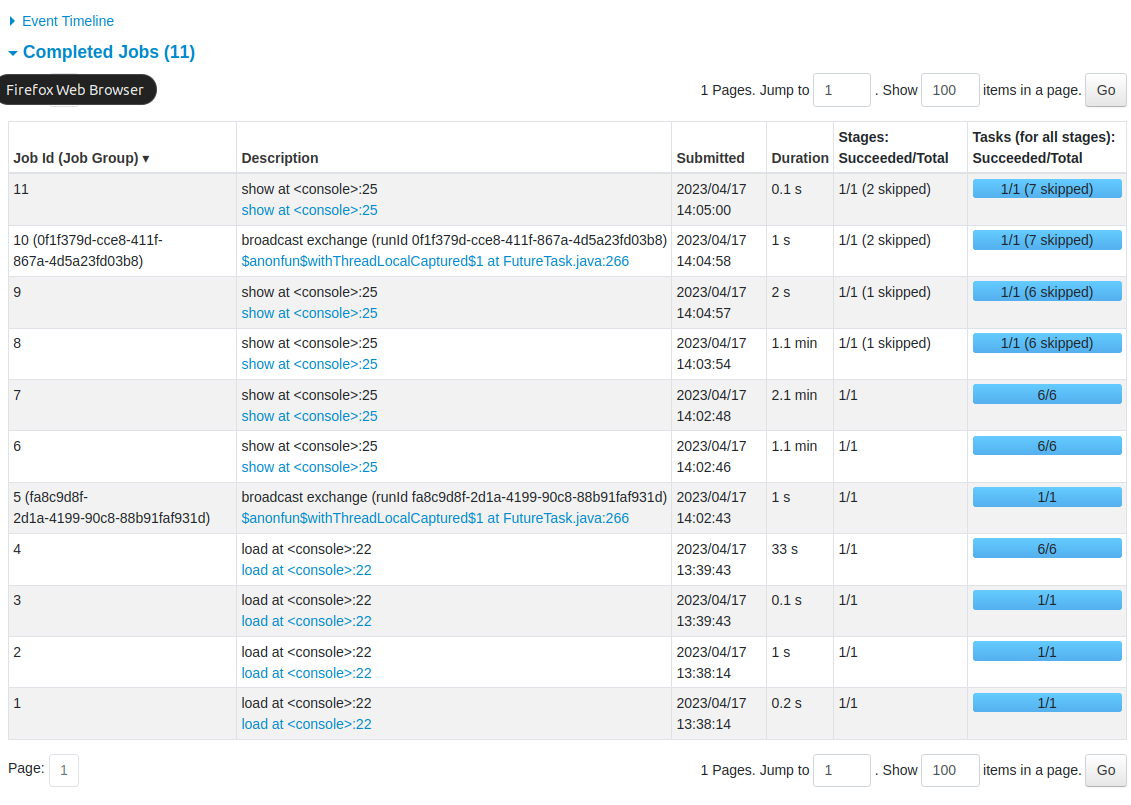
Q2. Provide the complete list of statements that produce the DataFrame that you can use for query. Use the Spark Web UI to view the timeline of spark processing in the Jobs page.

val movies = spark.read.format("csv").option("header", true).option("inferSchema", true).load("file:///usr/local/hadoop/spark-3.2.4-bin-without-hadoop/movies.csv")

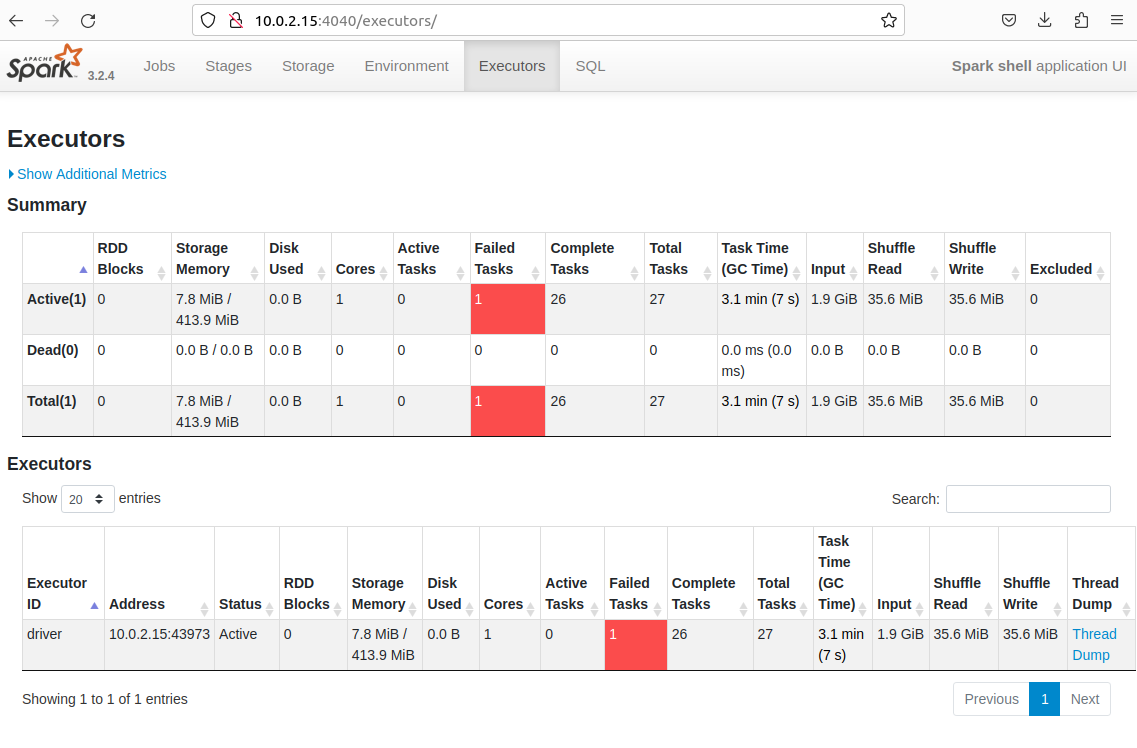
val ratings = spark.read.format("csv").option("header", true).option("inferSchema", true).load("file:///usr/local/hadoop/spark-3.2.4-bin-without-hadoop/ratings.csv")

val movie\_ratings = movies.join(ratings, "movieId").withColumn("genre\_list", split($"genres", "\\|")).select($"title", $"rating", explode($"genre\_list").as("genre"))

movie\_ratings.createOrReplaceTempView("movieRatingsView")



Q3. Take a screen shot of the Executors page show the metrics for your session.



### Query

Work through query statements to produce the answer to the analysis question.

Q4. Show the statements you used to produce the results.

val genres = spark.sql("select genre, title, count(\*) as rating\_count, avg(rating) avg\_rating from movieRatingsView group by genre, title order by genre, rating\_count desc")

val max\_genre = genres.groupBy($"genre".as("max\_genre")).agg(max("rating\_count").as("max\_rating\_count"))

genres.join(max\_genre, $"genre"===$"max\_genre" && $"rating\_count"===$"max\_rating\_count").drop("max\_genre").drop("max\_rating\_count")

Q5. Show the final results.

