Andrew Park (ap1614)
CS417: Distributed Systems
Spring 2021

# Project 2 Report: RUSpark

The implementation of the four programs is as follows:

1. **RedditPhotoImpact**: For **RedditPhotoImpact**, we first read the separate lines from the file and store it into a **JavaRDD**. Afterwards, we took lines and performed a **mapToPair**, mapping the **JavaRDD** to a **JavaPairRDD** with key-value pair type of **<Integer, Integer>** and with the **image_id** as the key and the impact(which is the sum of the comments, upvotes, and downvotes) as the value. Then, I called **reduceByKey** with the lambda function **(i1, i2) -> i1 + i2**. We then finished by calling **collect** and then printing the results.
2. **RedditHourImpact**: We first read and separated the lines from the file and stored it into a **JavaRDD**. Then, we took those lines and performed a **mapToPair**, mapping to a **JavaPairRDD** with key-value pair type of **<Integer, Integer>** and with the hours converted from Unix time to EST time(note EST, and not EDT, which may be confusing since we are currently in EDT) as keys and the impact as values. We then called **reduceByKey** using the same lambda function as before. We finish by calling **collect** and then printing our results.
3. **NetflixMovieAverage**: We first read and separated the lines from the file and stored it into a **JavaRDD**. We then took those lines and performed a **mapToPair** from the lines to a **JavaPairRDD** with a key-value pair type of **<Integer, Tuple2<Integer, Integer>>**, where the key is the **movie_id** and the value is a tuple of at **<1, rating>**. We then did **reduceByKey** with the lambda function **(i1, i2) -> new Tuple2<>(i1._1() + i1._2(),i2._1() + i2._2())**, which results in the Tuple pair **<# of ratings, sum of review ratings>**. We then call collect and iterate over each of the Tuple pairs. In each iteration, we compute the average rating and print the results in the desired format.
4. **NetflixGraphGenerate**: We first read and separated the lines from the file and stored it into a **JavaRDD**. We then used **mapToPair** to get a **JavaPairRDD** with key-value pairs **<Tuple2<Integer, Integer>,Integer>**, where the key is a Tuple containing **<movie_id, rating>** and with **customer_id** as the value. We then first join the **JavaPairRDD** with itself in order to create every possible pair of the Tuple **<customer_id, customer_id>** that have the same rating for the same movie. We then take this result and call **filter** with the lambda function **tuple -> tuple._2()._1() < tuple._2()._2()** in order to remove entries that are either of the form **<X,X>** or **<Y,X>**, where **X < Y**. This leaves us with a **JavaPairRDD** that has key values of **<movie_id, rating>** and a value of **<customer_id, customer_id>**, where each customer tuple pair is a pair of customers that reviewed the same movie with the same rating, indicating a similar taste. We then perform a **map** in order to take our **JavaPairRDD** and create a **JavaRDD** with type **Tuple2<Tuple2<customer_id, customer_id>, Tuple2<movie_id, rating>>** and take this **JavaRDD** and proceed to use **mapToPair** in order to create a JavaPairRDD using the lambda function **tuple -> new Tuple2<>(tuple._1(), 1)**. This leaves us with a JavaPairRDD that has key values of the **customer_id** pairs and a value of 1

initially(which is the similarity in taste). We then use reduceByKey with the lambda function **(i1, i2) -> i1 + i2** to sum up the similarities in taste. We then collect our results and then print out all the results.

For **RedditPhotoImpact**, we can run it with the **RedditData-Large.csv** and we see that the most impactful photo from the whole dataset is photo 1437, which had an impact of 192896.

For **RedditHourImpact**, we can run it with the **RedditData-Large.csv** and we see that the hour with the most impactful posts is 7PM, followed by 8PM, and then 9PM.

For **NetflixMovieAverage**, we can run it with the **NetflixData-Small.csv** and see that the movies with the highest average ratings are **The Rise and Fall of ECW** with an average rating of 3.92, **Dinosaur Planet** with an average rating of 3.75, and **Character** with an average rating of 3.64.

The hardest part of this project was figuring out how to properly do NetflixGraphGenerate. It was hard to figure out how to properly create the relations between the customer_id's. Without using join, it would have been difficult to figure out how to properly take the separate reviews and join them together based on customer ids and their reviews of each of the movies. By using join, I was able to create every combination of customer pairs that did have the same rating for the same movie. It helped that I read through the documentation and worked out what each of the different methods did through the documentation. Other than this issue, there were problems trying to launch Spark on AWS but this was due to an outdated NSS and CURL. Otherwise, most of the programs were straightforward and were almost identical to the WordCount example that was given as an example.There weren't many difficulties in this project that I found were problematic to a point where I was stuck on them. If anything, I would have read the documentation earlier since it would have helped me solve NetflixGraphGenerate much faster.