Master Informatique 2
Report for the Large Distributed Systems Project

Topic: Usage of Hadoop Map Reduce Usage for multiple task execution

Student: **Negruta Adrian**

Professor: **Fabrice Huet**

Summary

During this project, I undertook three primary tasks using two datasets that provide comprehensive information related to movies. These datasets detail the distribution of ratings from various users across multiple existing movies. The first objective was to identify the highest-rated movie ID for each user by leveraging the movie ratings dataset, which includes columns such as userID, movieID, and rating. Following this, the second task involved executing a join operation on the two datasets through a two-stage MapReduce process. This approach required the creation of an intermediate file to store the output of the first job, which was subsequently used as the input for the second job. The culmination of these efforts resulted in a final output that counts how many users liked each specific movie, effectively grouping them by like-count. For instance, an entry in the final file might appear as:

7281 Toy Story (1995)

This indicates that 7,281 users rated "Toy Story (1995)" favorably. Additionally, it is important to note that prior to processing the extensive provided datasets, I conducted tests on smaller subsets of the data. This preliminary testing was crucial to ensure efficient execution times and to address the significant constraints posed by the larger base datasets. In the subsequent sections of this report, I will delve into a more detailed explanation of the methodologies employed to achieve these results, complemented by screenshots from the execution process.

Table of Contents

1. The highest rated movieID per user

For this task, I developed a Hadoop MapReduce job on the ratings dataset in order to determine the highest-rated movie for each user.

To facilitate this, I created a custom Writable class, MovieRatingWritable, which encapsulates the movieID and rating fields, allowing efficient serialization and deserialization of these attributes during the MapReduce process. The mapper class, HighestRatedMoviePerUserMapper, processes each line of the input dataset by first skipping the header row. Then it parses the userID, movieID, and rating from each record and emits key-value pairs where the key is the userID and the value is an instance of custom class MovieRatingWritable containing the corresponding movieID and rating. By this, I ensure that during the shuffle and sort phase all movie ratings for a particular user are grouped together. The reducer class receives each userID along with an iterable of MovieRatingWritable objects. It iterates through these values to identify the movie with the highest rating for that user by comparing the rating values. Once the highest-rated movie is determined, the reducer emits the userID and the corresponding movieID of the top-rated movie.

Overall, this MapReduce job processes large-scale movie ratings data to extract meaningful insights about user preferences.

```
1	8327
2	30749
3	27773
```

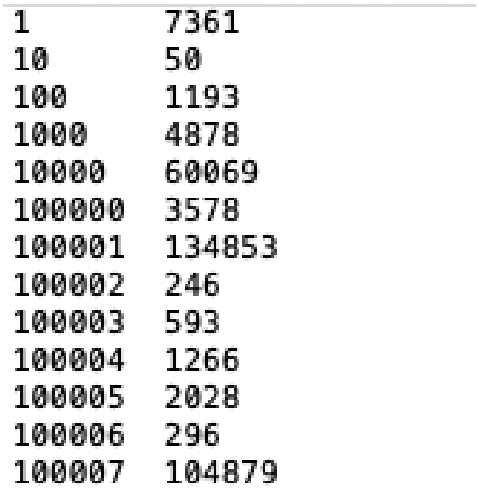
Figure 1 (represents for userID 1 – the highest rated movieID is 8327)

2. The highest rated movie name per userID

In this segment of the project, I implemented a two-stage Hadoop MapReduce workflow to determine the highest-rated movie names for each user.

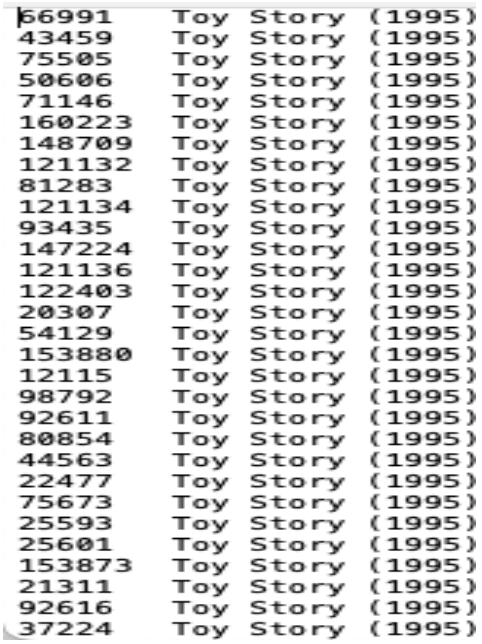
The first stage involves processing the ratings.csv dataset to identify the top-rated movie ID for every user. This is achieved through the RatingsMapper, which parses each record to extract userID, movieID, rating emitting key-value pairs with userID as the key and a combination of movieID and rating as the value. The corresponding MaxRatingReducer identifies the movie with the highest rating and emits the userID alongside the movieID of their favorite movie. An intermediate output from this stage serves as the input for the second MapReduce job.

The second stage focuses on enriching the data by joining the highest-rated movie IDs with their corresponding movie names from the movies.csv dataset. This action is orchestrated by JoinMovieNamesDriver, which utilizes the MultipleInputs feature to handle inputs from both the intermediate output and the movies.csv simultaneously. The JoinReducer performs the join operation by matching the movieIDs, associating each userID with the corresponding movie name. This results in a final output that lists each user alongside their highest-rated movie by name.



1	7361
10	50
100	1193
1000	4878
10000	60069
100000	3578
100001	134853
100002	246
100003	593
100004	1266
100005	2028
100006	296
100007	104879

Figure 2 – After job1 execution (intermediate file)



66991	Toy Story (1995)
43459	Toy Story (1995)
75505	Toy Story (1995)
50606	Toy Story (1995)
71146	Toy Story (1995)
160223	Toy Story (1995)
148709	Toy Story (1995)
121132	Toy Story (1995)
81283	Toy Story (1995)
121134	Toy Story (1995)
93435	Toy Story (1995)
147224	Toy Story (1995)
121136	Toy Story (1995)
122403	Toy Story (1995)
20307	Toy Story (1995)
54129	Toy Story (1995)
153880	Toy Story (1995)
12115	Toy Story (1995)
98792	Toy Story (1995)
92611	Toy Story (1995)
80854	Toy Story (1995)
44563	Toy Story (1995)
22477	Toy Story (1995)
75673	Toy Story (1995)
25593	Toy Story (1995)
25601	Toy Story (1995)
153873	Toy Story (1995)
21311	Toy Story (1995)
92616	Toy Story (1995)
37224	Toy Story (1995)

Figure 3 – After job2 execution (final output file)

3. Counter of how many users liked the movie, grouping by like-count

This task is divided into two stages as well as made for the second task. The purpose of this task was to analyze user preferences by counting the number of users who liked each movie and finally grouping movies based on their like-count.

The first stage contains the MovieNameMapper and MovieCountReducer classes. For each movie entry, the mapper emits a key-value pair of (MovieName, 1) by this, marking a single like for that movie. The reducer is in charge to aggregate these counts by summing the values associated with each movieName, resulting in a total like-count per movie. The intermediate output, server as the input for the second MapReduce job.

The second part involves in inverting the input data by emitting (LikeCount, MovieName) pairs, where LikeCount is the number of likes a movie has received. This inversion serves as an important action for the group by step based on their popularity. The GroupMoviesReducer then collects all movie names associated with each LikeCount, concatenating them into a single string. This will result in a final output where each record lists a like-count followed by the names of movies that share that count. For example, an output like “40 Toy Story Alien” indicates that both “Toy Story” and “Alien” received 40 likes from the users.

```
"Man with the Movie Camera, The (Chelovek s kino-apparatom) (1929)      1
"Manchurian Candidate, The (1962)       38
"Manchurian Candidate, The (2004)       1
"March of the Penguins (Marche de l'empereur, La) (2005)        3
"Maria Full of Grace (Maria, Llena eres de gracia) (2004)       1
"Mariachi, El (1992)    8
"Mark of Zorro, The (1940)      2
"Mask of Zorro, The (1998)      21
"Mask, The (1994)       73
"Master, The (2012)     6
"Matador, The (2005)    1
"MatchMaker, The (1997) 2
"Matrix Reloaded, The (2003)    30
"Matrix Revolutions, The (2003) 19
"Matrix, The (1999)     1277
"Matter of Life and Death, A (Stairway to Heaven) (1946)        3
"Maybe, Maybe Not (Bewegte Mann, Der) (1994)    1
"Maze Runner, The (2014)        6
"Me, Myself & Irene (2000)      7
"Mechanic, The (2011)   3
"Medallion, The (2003)  1
"Messenger: The Story of Joan of Arc, The (1999)        4
"Mexican, The (2001)    1
"Midnight Clear, A (1992)       1
"Mighty Ducks, The (1992)       2
"Mighty, The (1998)     3
"Milk of Sorrow, The (Teta asustada, La) (2009) 1
```

Figure 4 – After job1 execution (intermediate file)

```
656     Jurassic Park (1993)
705     "Lion King, The (1994)
726     Clueless (1995)
728     Terminator 2: Judgment Day (1991)
733     "Lord of the Rings: The Fellowship of the Ring, The (2001)
740     Clerks (1994)
753     Star Wars: Episode V - The Empire Strikes Back (1980)
754     Jumanji (1995)
801     Mr. Holland's Opus (1995) Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
836     Fargo (1996)
932     Fight Club (1999)
962     "City of Lost Children, The (Cité des enfants perdus, La) (1995)
986     GoldenEye (1995)
1034    Get Shorty (1995)
1064    "American President, The (1995)
1196    Dead Man Walking (1995)
1215    Blade Runner (1982)
1252    Casino (1995)
1277    "Matrix, The (1999)
1278    Léon: The Professional (a.k.a. The Professional) (Léon) (1994)
1406    "Silence of the Lambs, The (1991)
1436    Leaving Las Vegas (1995)
1855    Apollo 13 (1995)
1901    Babe (1995)
1961    "Godfather, The (1972)
1980    Taxi Driver (1976)
2041    Schindler's List (1993)
2660    Heat (1995)
2672    Sense and Sensibility (1995)
```

Figure 5 – After job2 execution (final file)

4. How to execute the project

4.0. Create jar file

- `mvn clean package` – to create the jar file which needs to be inserted in a shared workspace.

4.1. Run HighestRatedMoviePerUser

- Usage: `hadoop jar maven_dfhs-1.0-SNAPSHOT.jar mvn_dfhs.task1.HighestRateMoviePerUser /path_to_input_file /final_output_path`

  Exemple:

  ```
  hadoop jar maven_dfhs-1.0-SNAPSHOT.jar mvn_dfhs.task1.
  HighestRateMoviePerUser /input/ratings.txt /final_output
  ```

4.2. Run HighestRatedMovieName

1. *Job 1 (Find Highest-Rated Movie ID per User)*

- Usage: `hadoop jar maven_dfhs-1.0-SNAPSHOT.jar mvn_dfhs.task2.HighestRatedMovieIDDriver /path_to_input_ratings_file  /intermediate_output`

2. *Job 2 (Join Highest-Rated Movie IDs with Movie Names)*

- Usage: `hadoop jar maven_dfhs-1.0-SNAPSHOT.jar mvn_dfhs.task2.JoinMovieNamesDriver /intermediate_output /movies_input_file /final_output`

4.3. Run LikeCountByMovie

- Usage: `hadoop jar maven_dfhs-1.0-SNAPSHOT.jar mvn_dfhs.task3.LikeCountByMovie /path_to_input_file /path_to_intermediate_output /final_output_path`

  Exemple:

  ```
  Hadoop jar maven_dfhs-1.0-SNAPSHOT.jar
  mvn_dfhs.task3.LikeCountByMovie /input/user_movie_likes.txt
  /intermediate_output /final_output
  ```