UNIVERSITY of STIRLING

ITNPBD7 Assignment - Distributed Data Processing

MOVIE RATING ANALYSIS

Introduction

Analysis of movie reviews is a crucial component of the global entertainment industry, which produces a tremendous amount of data. It provides insight into how audiences respond to films, provides box office forecasts, and aids in enhancing marketing and promotion. A thorough summary of the analysis of a movie-review dataset with over 100,000 ratings from multiple reviewers in a variety of genres is provided in the study. The average rating for each film has been taken into account because it offers a broad quantitative summary of the audience's opinions; nonetheless, the average rating is only one of many elements considered when judging a film's quality. Lastly, each movie's average rating has been sorted according to the genre. The highest-ranking film in each genre has then been examined, which may help in the future to shed light on the key elements that go into making the highest-ranking movie as well as the potential impact that the director's methodology has on the ranking of movies.

Design

Movie reviews can be a significant source of textual data, which is exponentially growing and are often written in natural language, making them difficult to interpret and relevant for various applications, such as market & sentiment analysis and recommender systems. As a result, a storage system like HDFS, along with the MapReduce program on the Hadoop framework, is an excellent alternative because of its distributed structure, as it allows for the parallel processing of enormous datasets and is adaptable enough to be customised for various analyses compared to the regular file system. In addition, given the amount of data and the repetitive procedure of computational tasks involved in filtering out the best movies, MapReduce, a batch-processing framework, might be the preferred choice.

The following is a brief description of the gathered dataset before moving on to the data processing stage,

- The dataset comprises the Reviewer ID, Movie Title, Genre, Year of Release, and Rating.
- Around 100,000 reviews have been included in the analysis.
- A scale of 1 to 5 is used as a rating.
- The gathered dataset includes movies released between 1902 and 2018.

The analysis has been carried out using the MapReduce design, which divides the data into smaller portions and performs parallel processing. The partial aggregation of the mapper output has been carried out using a combiner in addition to the mapper and reducer, making it more effective and minimising computational costs. The following is an explanation of the data processing pipeline,

STEP 1: Mapper

After reading the input data, the mapper will categorise it into the following groups: ID, Movie Title, Genre, Year, and Rating. The information will subsequently be converted to lowercase, and all punctuation will be deleted. A years.txt file with a specific set of years has been utilised and processed using the loadYears function to analyse the top films during a particular era. The IF block produces output based on the years present in the years.txt file or if the file is empty; therefore, it will have varied results depending on whether the file contains any values. Since the mapper will only send the sorted data to the combiner/reducer, sorting the

movies based on years during the mapper phase may help reduce computational costs. Moreover, to avoid skewing the results and obtain an unbiased result, a threshold has been set to 1500(which is adjustable), eliminating reviews if the reviewer ID has been duplicated over 1500 times. These reviews are regarded to be the reviews rated by fans. The format of the output is (genre, title, float(rating), "1")), where the value 1 is the number of times each line appeared in the input data.

STEP 2: Combiner

The combiner receives input from the mapper and executes a partial aggregation to lessen the quantity of data delivered to the reducer node, assisting in **network traffic management**. In order to ensure the integrity of the data, the dictionary has also been **initialised with the tuple** (genre, title, and rating), as it is assumed to be a unique combination and would not change when the count value for each combination grows. In addition, the sum operation has been delegated to the combiner to aggregate the data and produce the same output as the mapper as well as to eliminate the possibility of inaccurate results or poor performance and increase scalability.

STEP 3: Reducer

The key-value pair produced by the combiner is processed in the reduce stage for the computational tasks. The films with fewer than 15 votes are eliminated at the beginning, and the pair of ratings and their count for each film are acquired and stored as a nested list in a dictionary. Subsequently, the average rating of each film is determined. In the final stage, the highest-ranked film of each genre was determined using the getMax() function.

In the mapper and combiner, the in-memory data structure **defaultdict()** [3] has been used in place of a regular dictionary because, when counting the frequency of words, a regular dict() checks each time to see if the key appears before increasing the count. In contrast, a defaultdict() avoids initialising a dictionary with a default value for each key, increasing efficiency, especially when handling large datasets. However, it has yet to be used in the reducer operations since it makes the unclear assumption that every key is present as well as it may take additional memory in the reducer job. Moreover, dictionaries have been adopted in all three stages to store intermediate results for efficient processing and lookups since they use hashing to look up and store the data.

Map/Reduce design without a Combiner:

In the aforementioned task, the combiner's role is to aggregate the output from the mapper and then pass the resulting key-value pair to the reducer. The job can still be completed even if we don't use a combiner, but because the output from the mapper will be sent directly to the reducer and the amount of data will be much larger, the cost of **network traffic and disc space** will go up, and the job may take longer. Additionally, there is a **strong likelihood of data skewness** as some keys generated from mapper output may have significantly higher values than others, which can be addressed by the partition tuning method [4]; some are combiners and partition algorithms. The default mechanism in a MapReduce job, hash partitioning, distributes the mapper output across the reducers [4]. Moreover**, data compression and load balancing** can be employed when combiners are impractical [5].

Improvement Suggestions:

Even though we are using a combiner and the aggregation functions are performed correctly by the reducer, the number of keys in the reducer function determines the number of reducers used in the design [11], where we have (genre, title) as the key and the combination of (genre, title) will be the number of reducers used, making the design a bit less efficient. It can be improved by finding the sum of ratings of each (genre, movie)

pair in the combiner itself, as well as printing the total count of ratings for that pair, where the combiner output will be **(genre, title, sum_of_rating, total_count)**. In this case, each (genre, movie) will be presented only once to the reducer, and utilising the sum and total count, the reducer can calculate the average rating of each movie and then one with the highest rating.

Current combiner output:
(genre, title, rating, total_count)
The total count, in this case, is the count of the combination (genre, title, rating), where the movie name will be printed repetitively because we are only counting similar ratings.

Modified combiner output:
(genre, title, sum_of_ratings, total_count)
Instead of individual ratings, we consider taking the sum of ratings for each movie in a genre as well as the total count of ratings to assist the reducer in determining the average rating of each movie. In this case, as we calculate the sum, the movie will be printed only once, resulting in fewer data sent to the reducer node but still producing output that is similar to that of the mapper, which will eventually minimize the number of reducers, assisting in cost efficiency and traffic management.

Use of HDFS

Given that the reviews we have compiled of movies range from 1902 to 2018 storing and processing all of them in a single machine would take time and effort. Since there is a significant amount of data to handle, HDFS has been preferred because it is a storage system that will store and replicate the data across multiple nodes rapidly, improving data throughput and ensuring that data can still be recovered from replicas even if a node fails, and making the system fault-tolerant, effective, and suitable for our requirements.

The stages through which HDFS processes data are listed below,

1. The input data is divided into chunks and distributed among nodes.
2. The data from each partition is then passed to the Mapper node, which generates the key-value pair stored in each node's local space.
3. Each mapper has a combiner at the end, which processes the mapper outcome in our case and produces an output equivalent to the mapper output after partial aggregation.
4. The combiner output is then sorted by key before being passed to the reducer node, where the reduce function performs the final aggregation and saves the output to the output file in HDFS.
5. For the fault tolerance (referenced in [10]), each node's files, including output files, are replicated across multiple nodes.

The process that has been followed to perform the data analysis on HDFS is given below,

To hold all the information, tasks, and outcomes associated with the review analysis, a directory has been created in Hadoop using the command **mkdir movie_ratings** in the initial step. And to configure the Hadoop map/reduce task, navigate to the movie_ratings directory using the command **cd ./movie_ratings/**.

```
rvr@vm021:~$ mkdir movie_ratings
rvr@vm021:~$ cd ./movie_ratings/
rvr@vm021:~/movie_ratings$
```

In the movie_ratings directory, all files—including data, mapper.py, combiner.py, reducer.py, runhadoop.sh, and simhadoop.sh—have been created. The tasks of the mapper, reducer and combiner have been detailed

in the upcoming sections. All files have been given read/write/execute permissions using the command **chmod u=rwx filename** since executing the files was restricted by a lack of permissions.

```
rvr@vm021:~/movie_ratings$ chmod u=rwx mapper.py
rvr@vm021:~/movie_ratings$ chmod u=rwx combiner.py
rvr@vm021:~/movie_ratings$ chmod u=rwx reducer.py
rvr@vm021:~/movie_ratings$ chmod u=rwx runhadoop.sh
rvr@vm021:~/movie_ratings$ chmod u=rwx simhadoop.sh
rvr@vm021:~/movie_ratings$
```

With the command **hdfs dfs -mkdir movie_ratings**, a directory has been created on HDFS since all job files have been created, and permission has been granted. And using **hdfs dfs -copyFromLocal ratings.txt movie_ratings**, the dataset has been copied to the new HDFS directory.

```
rvr@vm021:~/movie_ratings$ hdfs dfs -mkdir movie_ratings
rvr@vm021:~/movie_ratings$ hdfs dfs -ls
Found 3 items
drwxr-xr-x   - rvr ugtpg          0 2023-03-25 19:58 movie_ratings
drwxr-xr-x   - rvr ugtpg          0 2023-03-12 23:34 wordcount
drwx------   - rvr ugtpg          0 2023-02-17 11:14 yarn
rvr@vm021:~/movie_ratings$ hdfs dfs -copyFromLocal ratings.txt movie_ratings
rvr@vm021:~/movie_ratings$ hdfs dfs -ls movie_ratings
Found 1 items
-rw-r--r--   1 rvr ugtpg    5212892 2023-03-25 20:17 movie_ratings/ratings.txt
rvr@vm021:~/movie_ratings$
```

Using the shell script **./runhadoop.sh** file on Hadoop, we submit jobs the jobs on HDFS, which will process the data through mapper, combiner and then reducer, which fragment and parallelly process the data across the nodes on the cluster providing the results in a text file, which is then copied to the local Hadoop machine.

```
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=5216988
        File Output Format Counters
                Bytes Written=553
2023-03-25 20:36:44,407 INFO streaming.StreamJob: Output directory: /user/rvr/movie_ratings/results
rvr@vm021:~/movie_ratings$ hdfs dfs -ls movie_ratings
Found 2 items
-rw-r--r--   1 rvr ugtpg    5212892 2023-03-25 20:17 movie_ratings/ratings.txt
drwxr-xr-x   - rvr ugtpg          0 2023-03-25 20:36 movie_ratings/results
rvr@vm021:~/movie_ratings$ hdfs dfs -ls movie_ratings/results
Found 2 items
-rw-r--r--   1 rvr ugtpg          0 2023-03-25 20:36 movie_ratings/results/_SUCCESS
-rw-r--r--   1 rvr ugtpg        553 2023-03-25 20:36 movie_ratings/results/part-00000
rvr@vm021:~/movie_ratings$
```

```
rvr@vm021:~/movie_ratings$ hdfs dfs -cat movie_ratings/results/part-00000
action  army of darkness        5.0
adventure       army of darkness        5.0
animation       lion king the   4.71
children        lion king the   4.53
comedy  army of darkness        5.0
crime   shawshank redemption the        4.8
documentary     super size me   4.0
```

With the command **hdfs dfs -copyToLocal movie ratings/results/part-00000 results.txt**, the results gathered in the results/part-00000 have then been saved to the local Hadoop directory and analysed.

```
rvr@vm021:~/movie_ratings$ hdfs dfs -copyToLocal movie_ratings/results/part-00000 results.txt
rvr@vm021:~/movie_ratings$ more re
reducer.py   results.txt
rvr@vm021:~/movie_ratings$ more results.txt
action  army of darkness        5.0
adventure       army of darkness        5.0
animation       lion king the   4.71
children        lion king the   4.53
```

Analysis & Results

Two stages of analysis were carried out, the first of which involved filtering the movies based on the years listed in the years.txt file and the second of which involved analysing the entire dataset, producing two different results. In both results, Army of Darkness, a 1992 release, received the highest ratings in action, adventure, and horror, making it the highest-rated action-horror-comedy movie of all time. Combining these genres can produce an entertaining, humorous film that draws a large audience. Furthermore, Interstellar, one of my personal favourites, was observed to be on the top ratings in IMAX (both results) and as the best science fiction movie(with years filter), unquestionably because of its spectacular visuals with IMAX technology and scientific accuracy. Finally, a handful of genres, like adventure, fantasy, and documentaries, share the same films in both results, indicating that audiences seem more fascinated by these genres of movies in recent releases.

Distributed Computation

While MapReduce generally focuses on batch processing, Condor is a job scheduler designed for a broad range of tasks, including data analysis, machine learning, and batch processing operations that can be executed concurrently and can be dependent or independent [6]. Due to the absence of a data storage system, Condor depends on a storage system like HDFS, much like the Hadoop components, to perform the tasks. And Condor is appropriate for performing numerous independent tasks, often scientific calculations, and may distribute separate jobs across clusters. Nevertheless, the movie review analysis involves specific recurring operations where just one dataset needs to be processed and analysed, a task for which **Hadoop is suitable** even though Condor can handle it.

However, the processes that may be required for performing the same task in Condor may include,

- Create the mapper, combiner, and reducer in python(.py) and compressed it to an executable file named "**code_files.zip**".
- Create the **submission file (sub.txt)** with the following details,
  executable = code_files.zip

```
output = out.txt
error = out.err
log = out.log
requirements = OpSys == "LINUX"
transfer_input_files = code_files.zip
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
queue 1
```

- Create a directory and save or copy the submission file, the Python file " code files.zip ", and the dataset to it.
- Use the condor submit command **condor_submit sub.txt** after logging into the Condor.
- We may additionally use **condor q <username>** to check the job status.

Even though Hadoop is appropriate for movie rating research, some of its **drawbacks** include slow processing, a lack of support for stream processing(continuous input and output of data), extensive code, a lack of caching, which Spark may address, and finally, the fact that it only enables batch processing [9]. Finally, as the advanced stage of the analysis, Condor may be employed to build a model for a **movie recommender system** [7] that may be trained on the patterns and trends of movies & their reviews to provide personalised recommendations or a **trend analyser model**, such as the one described in [8], which may be employed to analyse the general trend of movies over time and forecast whether or not an upcoming film in a specific genre or directed by a specific person will be successful.

References

[1] https://docs.python.org/3/reference/expressions.html#membership-test-operations

[2] https://realpython.com/python-in-operator/

[3] https://docs.python.org/3/library/collections.html

[4] Y. Gao, Y. Zhou, B. Zhou, L. Shi, and J. Zhang, “Handling Data Skew in MapReduce Cluster by Using Partition Tuning,” vol. 2017, p. 1425102, 2017, doi: 10.1155/2017/1425102. [Online]. Available: https://doi.org/10.1155/2017/1425102

[5] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

[6] https://htcondor.readthedocs.io/en/latest/users-manual/welcome-to-htcondor.html

[7] http://hdl.handle.net/10415/4994

[8] P. A. Gloor, J. Krauss, S. Nann, K. Fischbach and D. Schoder, "Web Science 2.0: Identifying Trends through Semantic Social Network Analysis," 2009 International Conference on Computational Science and Engineering, Vancouver, BC, Canada, 2009, pp. 215-222, https://doi.org/10.1109/CSE.2009.186

[9] https://medium.com/@bharvi.vyas123/6-major-hadoop-limitations-with-their-solutions-1cae1d3936e1

[10] https://www.techtarget.com/searchdatamanagement/definition/Hadoop-Distributed-File-System-HDFS

[11] https://medium.com/edureka/mapreduce-tutorial-3d9535ddbe7c