IST3134 Big Data Analytics in the Cloud

Assignment

by

YEOH QI WEI

19046739

Bachelor of Information Systems (Honours) (Data Analytics)

Lecturer: Prof. Lau Sian Lun

Date: 7 August, 2023

Department of Computing and Information Systems

School of Engineering and Technology

Sunway University

Table of Contents

List of Figures

List of Tables

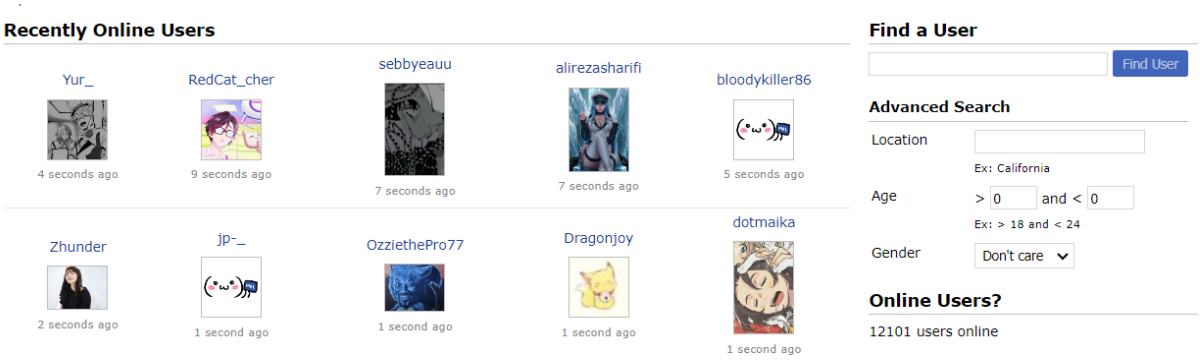
1.0 Problem Statement

The global popularity of Japanese animations (anime) and comics (manga) has been rising in recent years. Anime content encompasses diverse genres and can be in many forms. For people that want to search or discuss about anime content, they can go to a website called MyAnimeList. With millions of users, MyAnimeList (MAL) is a database and social networking site for the anime and manga community. The platform offers any information regarding anime or manga content, such as titles, sources, genres, synopses, voice actors, studios, and producers. The site also offers a space for users to engage in forums, clubs, and blogs to discuss about their interests. One feature of MAL is that it allows users to assign scores to anime and manga, reflecting their preferences and influencing the popularity of these titles. As such, the seasonal charts and popularity rankings reflect the consensus of the community.

The platform provides a few options for parties that would like to search for specific users. As shown in Figure 1, users can view at the real-time online users available on the site. Additional filters include usernames, locations, age range, and gender. However, the platform currently lacks a mechanism to discover users based on their activity within the community. For example, it might be difficult for entities that would want to reach out to active users on the platform. This includes advertisers, researchers, content creators or distributors, event planners, or any parties that may want to interact with the community on MAL.

As a result, in this assignment, we would like to find a way to identify and acknowledge users who actively contribute on MAL by providing ratings that genuinely represent the collective opinion of the community.

Figure 1: *Current implementation of the search function for users in MAL*



2.0 Dataset

The dataset selected for this project is the 4.4GB large comma-separated values file, final_animedataset.csv generated by Uddin (2023) on Kaggle. The dataset was generated by Uddin using the 2018 data from MyAnimeList using the Jikan API. The dataset consists of user information, anime information, and user rating on the anime. Table 1 shows more information on each column in the dataset.

The dataset allowed us to find the top 10 users with a highest number of valid ratings. A valid rating was defined as the user's rating of the anime that represents the opinion of the community. To obtain this, a user's rating of an anime (*my_score*) was examined if it was within ±1 from the average score of the anime given by the community (*score*). In addition, animes that were not scored by at least 1000 users were not considered as sample size was small, thus could be easily affected by new scores that deviates from the mean.

Dataset link: https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset

Table 1: *Explanation of each item in the final_animedataset.csv*

No	Column	Information
1	username	Username of user that provided rating.
2	anime_id	Unique identifier of anime title.
3	my_score	Rating given by user for the anime on a scale of 1 to 10.
4	user_id	Unique identifier of user.
5	Gender	Gender of user.
6	Title	Title of anime.
7	Type	Type of anime. (TV series, movie, specials, OVA, etc)
8	source	Source material of anime. (Light novel, web novel, manga, etc)
9	score	Average rating score of the anime based on all user ratings.
10	scored_by	Number of users who have rated the anime.
11	rank	Ranking of anime.
12	popularity	Popularity rank of anime.
13	genre	Genre(s) of anime.

3.0 Methods

In this assignment, we used the MapReduce approach to efficiently identify the top 10 users who actively contribute on MAL by providing ratings that genuinely represent the collective opinion of the community. In combination with Hadoop clusters, the MapReduce framework was selected as it enabled distributed processing of large datasets like the final_animedataset.csv. The dataset would be split into blocks and distributed across the nodes on a Hadoop cluster. The master node would assign MapReduce jobs to the slave nodes, allowing the cluster to process the dataset parallelly. In this project, we tested the performance of MapReduce on a local node and on a Hadoop cluster with 5 nodes (1 master node and 4 slave nodes).

To create a baseline for performance, the task was conducted using a conventional approach in Python with dictionaries and loopers. Furthermore, as the nature of the problem was related to querying, we also applied Apache Hive to perform the task and compared their performance.

3.1 MapReduce

3.1.1 Mapper

The goal of the Mapper was to extract and filter relevant data from the input dataset. In our implementation, the Mapper took a set of records representing ratings made by users and filtered out user ratings that fall within a specific range from the average score of the anime. The range considered was ±1 from the average score of the anime. Additionally, we only considered ratings made on animes that have at least been scored by 1000 users. The Mapper then output the values *user_id*, *username*, and a constant value of 1. The three values were separated by a tab.

3.1.2 Sorting Phase

Between the Mapper and Reducer, the values generated by the Mapper were sorted.

3.1.3 Reducer

The Reducer was responsible for aggregating and summarizing the data generated by the Mapper. In the assignment, the Reducer received three values from the Mapper and converted them into key-pair values by splitting the input based on the last tab. As such, the key would be “user_id\tusername”, representing a user, and the value would be “1”.

The Reducer would iterate through the sorted key-pair values and aggregate the count of 1s for each user. It would keep track of the current user being processed and the count of

their valid ratings. Whenever a new user was encountered, the Reducer would output the aggregated count for the previous user and proceed with the next user. The output of the Reducer was a list of users along with the total count of their valid ratings within the specified range.

3.1.4 Final Output

Once the output list was generated by the Reducer, we would finally sort the list based on the number of valid ratings and display the top 10 users with the highest number of valid ratings, along with their user IDs.

3.2 Apache Hive

Apache Hive is a data warehousing and SQL-like query language built on top of Apache Hadoop. It allows users to perform SQL-based queries on large-scale datasets stored in Hadoop Distributed File System (HDFS), making it well-suited for big data processing.

Firstly, the dataset was stored in a table called 'ratings' for querying purposes in Apache Hive. From the 'ratings' table, we filtered out the ratings to include only those where the user's own rating (*my_score*) falls within a range of ±1 from the average score (*score*). This ensured that we were considering only ratings that are close to the average rating, which best represents the consensus of the community. In addition, we would only consider ratings done on anime that was scored by at least 1000 users. Next, we would count the remaining records after the filter and aggregate them by grouping the filtered records based on *user_id* and *username*. Similar to the MapReduce approach, the final result of the query would contain three columns: *user_id*, *username*, and *count*.

Since Apache Hive operates on top of the Hadoop cluster, the input data would be divided into multiple blocks and be processed parallelly. As such, the query task was processed independently across multiple blocks, creating multiple output files to store the partial results. Thus, for the final display, the output files were concatenated into a single final output file.

Link to scripts and output files in GitHub: https://github.com/SaltQW/IST3134-19046739

4.0 Results and Discussion

In Linux, the `time` command displays three types of time, namely *real*, *user*, and *sys*. *Real* refers the real time taken to carry out the entire execution of a process. It provides a more comprehensive view of the overall time taken to complete a process, including the time spent executing code and the time spent waiting for external dependencies. *User* shows the amount of CPU time spent by the process at the regular user-level applications, including user-generated code and excluding kernel-level operations. *Sys* shows the amount of CPU time spent by the process at the kernel-level operations, including system calls and network communication.

As Hadoop distribute the tasks across multiple nodes in a cluster, the system does not reflect the correct time taken by the task using the time shown by *user* and *sys*. Time used by additional activities by the Hadoop resource manager like scheduling overhead, context switching, and coordination between tasks may not show up in *user* and *sys*. As such, all time measurement in this section was taken from the more realistic representation for time taken by the process, the *real* section generated by the `time` command.

As the time taken by the process may vary, certain tasks that had closer times were repeated to obtain a more stable average time. Table 2 shows the results of the methods that was tested 3 times and their respective average time taken to execute the process.

The display of results in Table 3 refers to sorting the users based on the highest total count of valid ratings and displaying the top 10 users with their user ID. As the Apache Hive method produced multiple partial output files, an additional step of concatenation was executed before displaying the results. Thus, the time taken by the Apache Hive method (2.825 seconds) in Table 3 was calculated by adding the time taken to concatenate the files (2.562 seconds) and the time taken to display the results (0.263 seconds).

Table 2: *Average time taken by method*

Method	Run 1	Run 2	Run 3	Average
MapReduce on local machine	1 minute 36.311seconds	1 minute 39.792 seconds	1 minute 36.036 seconds	1 minute 37.380 seconds
MapReduce on Hadoop Cluster	1 minute 28.895 seconds	1 minute 26.013 seconds	1 minute 26.651 seconds	1 minute 27.186 seconds
Apache Hive on Hadoop Cluster	80.201 seconds	71.446 seconds	76.853 seconds	1 minute 16.167 seconds

Table 3: *Time taken by all methods*

Method	**Process**	**Display**	**Total**
Conventional	3 minutes 13.391 seconds	0.260 seconds	3 minutes 13.651 seconds
MapReduce on local machine	1 minute 37.380 seconds	0.217 seconds	1 minute 37.597 seconds
MapReduce on Hadoop Cluster	1 minute 27.186 seconds	2.787 seconds	1 minute 29.973 seconds
Apache Hive on Hadoop Cluster	1 minute 16.167 seconds	2.825 seconds	1 minute 18.992 seconds

The display of results in Table 3 refers to sorting the users based on the highest total count of valid ratings and displaying the top 10 users with their user ID. As the Apache Hive method produced multiple partial output files, an additional step of concatenation was executed before displaying the results. Thus, the time taken by the Apache Hive method (2.825 seconds) in Table 3 was calculated by adding the time taken to concatenate the files (2.562 seconds) and the time taken to display the results (0.263 seconds).

Based on the results in Table 3, the MapReduce approaches were significantly faster than the conventional approach in Python. The conventional method had doubled the time taken for the MapReduce processes to complete the task. Between running the MapReduce task on a local node and on the Hadoop cluster, it was faster to complete the task on the Hadoop cluster. This demonstrated the processing speed of MapReduce on a Hadoop cluster when processing large datasets.

Based on the results, it was observed that Apache Hive outperformed MapReduce when run on a Hadoop cluster. This could be influenced by the query optimiser in Apache Hive. Instead of manually creating a MapReduce script, the optimiser finds the most optimal way to execute the query given, resulting in a faster performance. It was noted that methods that involved Hadoop cluster took longer times to display the results. This may be caused by the need to transfer data across the network from nodes in the cluster to the node that requested it.

5.0 Reflection

The results have shown that the MapReduce framework is a significantly faster way to process large datasets. Moreover, although Apache Hive queries are faster than MapReduce jobs, Apache Hive can only perform tasks associated with querying while MapReduce has better flexibility and control for custom data processing tasks. Therefore, it was concluded that users should prioritise Apache Hive when the task is related to querying and employ MapReduce on Hadoop when processing large datasets.

In addition, during the early stages in the project, it was observed that running MapReduce jobs on a Hadoop cluster with 2 slave nodes was significantly slower than simply running the task on a local node. It demonstrated that although Hadoop clusters could have a higher processing power, Hadoop clusters that are too small may not be able to perform the task with less time. This was likely due to the additional overheads such as communication and task allocation between the nodes within the cluster. As such, we must consider the size of the cluster when deciding on using Hadoop to process large datasets.

The project has changed my approach to data processing, especially when dealing with large datasets. I have accustomed to the conventional methods when processing data in my personal projects. From this assignment, I have learned to consider the MapReduce framework, usage of Hadoop, and Apache Hive when handling large datasets.

References

Uddin, S. (2023, June). Anime Dataset. Retrieved July 25, 2023, from https://www.kaggle.com/datasets/dbdmobile/myanimelist-dataset

Appendix A

Conventional method

Time taken by the process of conventional method.

```
hadoop@ip-172-31-81-89:~/workspace/python$ time python3 ~/workspace/python/valid_rating_count.py
/home/hadoop/final_animedataset.csv > ~/workspace/base_output

real    3m13.391s
user    3m8.898s
sys     0m4.446s
```

Time taken to display the output of conventional method.

```
hadoop@ip-172-31-81-89:~/workspace/python$ time cat ~/workspace/base_output | awk -F'\t'
 '{print $3 "\t" $0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
8669    spacecowboy     3972
123582  Shouichirou     2943
74773   ShanaFlame      2882
2007638 MaouSatan       2757
5585853 Garyus  2573
1237755 DeadlyKizuna    2552
305748  BerryChanx      2526
15307   ziomalos        2367
6825    edinaa  2252
291713  Dedzapadlo      2232

real    0m0.260s
user    0m0.252s
sys     0m0.023s
```

Appendix B

MapReduce on local machine

3 Samples of time taken by the process of MapReduce on a local machine.

```
hadoop@ip-172-31-81-89:~/workspace/python$ time cat /home/hadoop/final_animedataset.csv
| python3 /home/hadoop/workspace/python/mapper.py | sort - | python3 /home/hadoop/worksp
ace/python/reducer.py > /home/hadoop/workspace/output

real    1m36.311s
user    1m35.836s
sys     0m13.969s
```

```
hadoop@ip-172-31-81-89:~/workspace/python$ time cat /home/hadoop/final_animedataset.csv
| python3 /home/hadoop/workspace/python/mapper.py | sort - | python3 /home/hadoop/worksp
ace/python/reducer.py > /home/hadoop/workspace/output

real    1m39.792s
user    1m38.954s
sys     0m14.300s
```

```
hadoop@ip-172-31-81-89:~/workspace/python$ time cat /home/hadoop/final_animedataset.csv
 | python3 /home/hadoop/workspace/python/mapper.py | sort - | python3 /home/hadoop/work
space/python/reducer.py > /home/hadoop/workspace/output

real    1m36.036s
user    1m35.345s
sys     0m13.934s
```

Time taken to display the output of MapReduce on local machine.

```
hadoop@ip-172-31-81-89:~/workspace/python$ time cat ~/workspace/output | awk -F'\t'
'{print $3 "\t" $0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
8669    spacecowboy     3919
123582  Shouichirou     2891
74773   ShanaFlame      2842
2007638 MaouSatan       2717
5585853 Garyus  2534
1237755 DeadlyKizuna    2512
305748  BerryChanx      2486
15307   ziomalos        2334
6825    edinaa  2220
291713  Dedzapadlo      2210

real    0m0.217s
user    0m0.217s
sys     0m0.019s
```

Appendix C

MapReduce on Hadoop Cluster

3 Samples of time taken by the process of MapReduce on the Hadoop cluster.

```
2023-08-03 10:54:07,208 INFO streaming.StreamJob: Output directory: /output

real    1m28.895s
user    0m5.751s
sys     0m0.465s
```

```
2023-08-03 11:08:22,519 INFO streaming.StreamJob: Output directory: /output

real    1m26.013s
user    0m5.878s
sys     0m0.406s
```

```
2023-08-03 11:11:46,067 INFO streaming.StreamJob: Output directory: /output

real    1m26.651s
user    0m5.870s
sys     0m0.426s
```

Time taken to display the output of MapReduce on the Hadoop cluster.

```
hadoop@ip-172-31-81-89:~/workspace/python$ time hadoop fs -cat /output/part-00000 |
 awk -F'\t' '{print $3 "\t" $0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
8669    spacecowboy     3919
123582  Shouichirou     2891
74773   ShanaFlame      2842
2007638 MaouSatan       2717
5585853 Garyus  2534
1237755 DeadlyKizuna    2512
305748  BerryChanx      2486
15307   ziomalos        2334
6825    edinaa  2220
291713  Dedzapadlo      2210

real    0m2.787s
user    0m3.571s
sys     0m0.276s
```

Appendix D

Apache Hive on Hadoop Cluster

3 Samples of time taken by the process of Apache Hive on the Hadoop cluster.

```
MapReduce Total cumulative CPU time: 3 minutes 15 seconds 290 msec
Ended Job = job_1691059695805_0004
Moving data to directory /hive_output
MapReduce Jobs Launched:
Stage-Stage-1: Map: 17  Reduce: 18   Cumulative CPU: 195.29 sec   HDFS Read: 4550273652 HDFS Write: 2259430 SUCCESS
Total MapReduce CPU Time Spent: 3 minutes 15 seconds 290 msec
OK
Time taken: 80.201 seconds
```

```
MapReduce Total cumulative CPU time: 3 minutes 16 seconds 400 msec
Ended Job = job_1691059695805_0005
Moving data to directory /hive_output
MapReduce Jobs Launched:
Stage-Stage-1: Map: 17  Reduce: 18   Cumulative CPU: 196.4 sec   HDFS Read: 4550273788 HDFS Write: 2259430 SUCCESS
Total MapReduce CPU Time Spent: 3 minutes 16 seconds 400 msec
OK
Time taken: 71.446 seconds
```

```
MapReduce Total cumulative CPU time: 3 minutes 18 seconds 10 msec
Ended Job = job_1691059695805_0006
Moving data to directory /hive_output
MapReduce Jobs Launched:
Stage-Stage-1: Map: 17  Reduce: 18   Cumulative CPU: 198.01 sec   HDFS Read: 4550273788 HDFS Write: 2259430 SUCCESS
Total MapReduce CPU Time Spent: 3 minutes 18 seconds 10 msec
OK
Time taken: 76.853 seconds
```

Time taken to concatenate the partial results.

```
hadoop@ip-172-31-81-89:~/workspace$ time hadoop fs -getmerge /hive_output ~/workspace/hive_output

real    0m2.562s
user    0m3.323s
sys     0m0.256s
```

Time taken to display the output of Apache Hive on the Hadoop cluster.

```
hadoop@ip-172-31-81-89:~/workspace$ time cat hive_output | awk -F'\t'
'{print $3 "\t" $0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
8669    spacecowboy     3919
123582  Shouichirou     2891
74773   ShanaFlame      2842
2007638 MaouSatan       2717
5585853 Garyus  2534
1237755 DeadlyKizuna    2512
305748  BerryChanx      2486
15307   ziomalos        2334
6825    edinaa  2220
291713  Dedzapadlo      2210

real    0m0.263s
user    0m0.265s
sys     0m0.018s
```

Appendix E

Log of all key commands in this assignment

- Created a S3 Bucket to upload and store the dataset (final_animedataset.csv).
- Setup and created 5 EC2 instances using the SunU-Hadoop-Image v1.4 AMI (Keypair, Security Group, Instance Type (t2.medium), IAM role (LabInstanceProfile))
- Add all private IP addresses of master nodes and 4 slave nodes in all nodes' host file
  ```
  sudo nano /etc/hosts
  ```
- Change user to hadoop
  ```
  sudo su - hadoop
  ```
- Add "slave3" and "slave4" in the workers file for the configuration of Hadoop cluster.
  ```
  nano hadoop-3.3.6/etc/hadoop/workers (add slave3, slave4)
  ```

Setup in master node

- Install AWS Command Line Interface to access the S3 Bucket
  ```
  sudo apt install awscli
  ```
- Change user to Hadoop
- Download the dataset to the local machine (master node)
  ```
  aws s3 sync s3://ist3134-animedata /home/Hadoop
  ```
- Format the namenodes
  ```
  hdfs namenode -format
  ```
- Starts all Hadoop daemons, the namenode, datanodes
  ```
  start-all.sh
  ```
- Upload dataset to Hadoop cluster
  ```
  hadoop fs -put final_animedataset.csv /
  ```
- Create directory for the scripts on local machine
  ```
  mkdir ~/workspace
  mkdir ~/workspace/python
  ```
- Change directory
  ```
  cd workspace/python
  ```

Conventional Method

- Create python script.
  ```
  nano valid_rating_count.py
  ```

- Change the permission of the script to be executable.

```
chmod +x valid_rating_count.py
```

- Run the script.

```
time python3 ~/workspace/python/valid_rating_count.py
/home/hadoop/final_animedataset.csv > ~/workspace/base_output
```

- Display output. (Copy the count column to the first column, sort the data, delete the first column, and show the top 10 results.)

```
time cat ~/workspace/base_output | awk -F'\t' '{print $3 "\t"
$0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
```

MapReduce in local node

- Create MapReduce Python scripts.

```
nano mapper.py
nano reducer.py
```

- Run the script. (Mapper, sort, Reducer)

```
time cat /home/hadoop/final_animedataset.csv | python3
/home/hadoop/workspace/python/mapper.py | sort - | python3
/home/hadoop/workspace/python/reducer.py >
/home/hadoop/workspace/output
```

- Display output.

```
time cat ~/workspace/output | awk -F'\t' '{print $3 "\t" $0}'
| sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
```

MapReduce on Hadoop Cluster

- Run the script.

```
time hadoop jar /home/hadoop/hadoop-
3.2.2/share/hadoop/tools/lib/hadoop-streaming-3.2.2.jar -input
/final_animedataset.csv -output /output -file mapper.py -file
reducer.py -mapper mapper.py -reducer reducer.py
```

- Display output.

```
time hadoop fs -cat /output/part-00000 | awk -F'\t' '{print $3
"\t" $0}' | sort -t$'\t' -k1,1nr | cut -f 2- | head -n 10
```

Apache Hive on Hadoop Cluster

- Initialize the Hive Metastore schema using Apache Derby as the database and enter Apache Hive.

```
schematool -initSchema -dbType derby
hive
```

- Create a table to store dataset before processing.

```
CREATE TABLE ratings (
username string, anime_id string, my_score float, user_id
string, gender string, title string, type string, source
string, score float, scored_by int, rank string, popularity
string, genre string)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
TBLPROPERTIES ("skip.header.line.count"="1");
```

- Load dataset into the table.

```
LOAD DATA LOCAL INPATH '/home/hadoop/final_animedataset.csv'
INTO TABLE ratings;
```

- Run the Hive query and store output onto Hadoop cluster.

```
INSERT OVERWRITE DIRECTORY '/hive_output'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
SELECT user_id, username, COUNT(*) AS count
FROM ratings
WHERE my_score BETWEEN (score - 1) AND (score + 1)
AND scored_by >= 1000
GROUP BY user_id, username;
```

- Leave Apache Hive

```
exit;
```

- Concatenate the partial outputs in Hadoop cluster

```
time hadoop fs -getmerge /hive_output ~/workspace/hive_output
```

- Display output.

```
time cat hive_output | awk -F'\t' '{print $3 "\t" $0}' | sort
-t$'\t' -k1,1nr | cut -f 2- | head -n 10
```