Submitted By:

Abhishek Duggirala (1002031522)

OVERALL STATUS:

Hadoop makes it easier to use multiple computers together to work on really big amounts of data. It helps to efficiently store and process large datasets by breaking them into smaller pieces and distributing them across different servers. This allows for faster processing and better utilization of resources.

After executing the necessary steps to process the IMDB data, we are pleased to report that we have been successful in generating sets of output, fulfilling the requirements of the problem description. The processing involved the use of Hadoop's Map/Reduce paradigm to efficiently handle the large dataset, allowing for parallel processing on a cluster of servers.

Major components that we implemented in this project:

1. Install wsl
2. Install Java: Hadoop requires Java to run. Install Java on your WSL system by running the following command in the terminal

sudo apt-get install openjdk-8-jdk

1. Download Hadoop tar file from Apache website. Extract Once the download is complete, extract the tar file .

wget https://downloads.apache.org/hadoop/common/hadoop-3.3.2/hadoop-3.3.2.tar.gz

1. Set environment variables: Set the following environment variables in your .bashrc or .bash\_profile file:

export HADOOP\_HOME=/home/abhishek/hadoop-3.3.2

1. Configure Hadoop: Navigate to the /hime/abhishek/Hadoop-3.3.2/etc/hadoop directory and edit the core-site.xml and hdfs-site.xml files to configure Hadoop.
2. To be able to use Hadoop, it is necessary to format the HDFS file system first. You can format the HDFS by running the command provided below:

--- hdfs namenode -format

1. Now you can start using Hadoop on WSL by running the start-all.sh script in the /home/abhishek/Hadoop-3.3.2/sbin directory.

To address the problems assigned, we only had to create and utilize two separate Java methods, each designed for a specific task. One is Mapper method and the other one is Reducer method.

Using the Mapper method, input data is processed and key-value pairs are generated. Input data is received as records, each of which contains a key-value pair. These records are processed by the Mapper method, which generates intermediate key-value pairs that are then passed to the Reducer method.

The Reducer method, on the other hand, receives the intermediate key-value pairs generated by the Mapper method, and performs further processing on them to produce the final output. The Reducer method combines the values associated with each intermediate key to generate the final output.

These two methods enable Hadoop to process and analyze large datasets in a distributed manner. By breaking down the data processing task into smaller chunks and distributing it across the cluster, Hadoop can efficiently handle large-scale data processing tasks that would otherwise be infeasible on a single machine. But, in our project we have done the work using single node cluster.

FILE DESCRIPTION:

For easier testing we used a test file which contains 10 records to see if the functionality is working as expected.

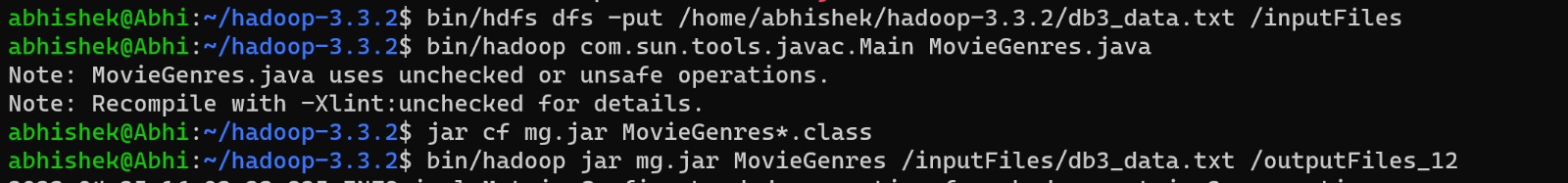
LOGICAL ERRORS :

1. While performing setup phase using virtual machine we encountered errors, then we tried Hadoop using wsl then everything worked fine.
2. At first, we created key using StringBuffer while appending genres it didn’t append properly. Then we took each genre into a list and compared the genres list separately.
3. While executing we created a separate directory for this project, but the jar file was in the home directory of hadoop and class files were in other directory there we faced a problem.

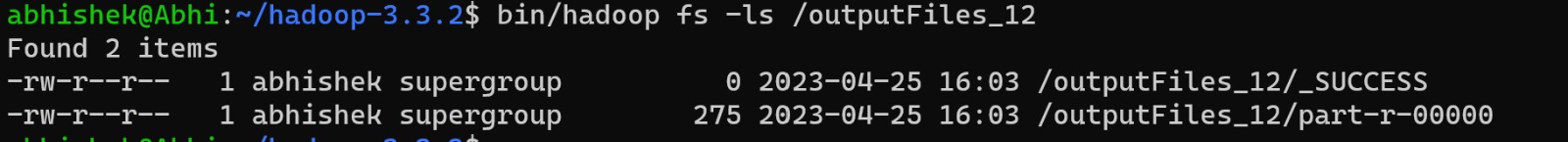
OUTPUT AND ANALYSIS:

Task 1:

Following commands are used to implement the task



In below screen shot we can see that the output is in outputFiles



This is the final output

Text

Description automatically generated with medium confidence

ANALYSIS:

From below screenshot we can see that

* Year on year content produced has been increased drastically.
* Adventure and Sci-Fi type of movies are produced very less compared to other gernes.
* The percentage of content increased is almost the same in both Action-Drama and Comedy-Romance gernes.

Chart, bar chart

Description automatically generated

Output:

SQL> SELECT A.PRIMARYTITLE,A.TITLETYPE, B.NUMVOTES, B.AVERAGERATING FROM imdb00.title\_basics A JOIN imdb00.title\_ratings B ON A.TCONST=B.TCONST WHERE B.NUMVOTES>=100000 AND A.TITLETYPE IN('movie','tvMovie') AND A.STARTYEAR BETWEEN 2000 AND 2006 AND GENRES LIKE '%Comedy%' and GENRES LIKE '%Romance%' ORDER BY B.AVERAGERATING DESC FETCH FIRST 5 ROWS ONLY;

PRIMARYTITLE

--------------------------------------------------------------------------------

TITLETYPE

--------------------------------------------------------------------------------

NUMVOTES AVERAGERATING

---------- -------------

Amelie

movie

748048 8.3

Good Bye Lenin!

movie

145590 7.7

PRIMARYTITLE

--------------------------------------------------------------------------------

TITLETYPE

--------------------------------------------------------------------------------

NUMVOTES AVERAGERATING

---------- -------------

Love Actually

movie

476576 7.6

Sideways

movie

PRIMARYTITLE

--------------------------------------------------------------------------------

TITLETYPE

--------------------------------------------------------------------------------

NUMVOTES AVERAGERATING

---------- -------------

190672 7.5

Garden State

movie

216016 7.4

Analysis:

SQL> EXPLAIN PLAN FOR SELECT A.PRIMARYTITLE,A.TITLETYPE, B.NUMVOTES, B.AVERAGERATING FROM imdb00.title\_basics A JOIN imdb00.title\_ratings B ON A.TCONST=B.TCONST WHERE B.NUMVOTES>=100000 AND A.TITLETYPE IN('movie','tvMovie') AND A.STARTYEAR BETWEEN 2000 AND 2006 AND GENRES LIKE '%Comedy%' and GENRES LIKE '%Romance%' ORDER BY B.AVERAGERATING DESC FETCH FIRST 5 ROWS ONLY;

Explained.

SQL> --Printing the plan

SQL> SELECT \* FROM TABLE(DBMS\_XPLAN.DISPLAY);

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

Plan hash value: 2653010624

--------------------------------------------------------------------------------

----------------

| Id | Operation | Name | Rows | Bytes | Cost (%

CPU)| Time |

--------------------------------------------------------------------------------

----------------

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 0 | SELECT STATEMENT | | 5 | 10280 | 5228

(1)| 00:00:01 |

|\* 1 | VIEW | | 5 | 10280 | 5228

(1)| 00:00:01 |

|\* 2 | WINDOW SORT PUSHED RANK | | 5 | 575 | 5228

(1)| 00:00:01 |

| 3 | NESTED LOOPS | | 5 | 575 | 5227

(1)| 00:00:01 |

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

| 4 | NESTED LOOPS | | 2071 | 575 | 5227

(1)| 00:00:01 |

|\* 5 | TABLE ACCESS FULL | TITLE\_RATINGS | 2071 | 35207 | 1084

(2)| 00:00:01 |

|\* 6 | INDEX UNIQUE SCAN | SYS\_C00547784 | 1 | | 1

(0)| 00:00:01 |

|\* 7 | TABLE ACCESS BY INDEX ROWID| TITLE\_BASICS | 1 | 98 | 2

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

(0)| 00:00:01 |

--------------------------------------------------------------------------------

----------------

Predicate Information (identified by operation id):

---------------------------------------------------

1 - filter("from$\_subquery$\_004"."rowlimit\_$$\_rownumber"<=5)

2 - filter(ROW\_NUMBER() OVER ( ORDER BY INTERNAL\_FUNCTION("B"."AVERAGERATING"

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

) DESC

)<=5)

5 - filter("B"."NUMVOTES">=100000)

6 - access("A"."TCONST"="B"."TCONST")

7 - filter("A"."GENRES" LIKE U'%Comedy%' AND "A"."GENRES" LIKE U'%Romance%' A

ND

("A"."TITLETYPE"=U'movie' OR "A"."TITLETYPE"=U'tvMovie') AND

TO\_NUMBER("A"."STARTYEAR")>=2000 AND TO\_NUMBER("A"."STARTYEAR")<=2

006 AND "A"."GENRES"

PLAN\_TABLE\_OUTPUT

--------------------------------------------------------------------------------

IS NOT NULL AND "A"."GENRES" IS NOT NULL)

Note

-----

- this is an adaptive plan