**DBMS Models and implementation (CSE 5331)**

Project III:

**Map/Reduce Programming Exercise and Analysis**

# SUBMITTED BY: SAHIBA KHAN (UTA ID: 1002083293)

# INSTRUCTOR: **Sharma Chakravarthy**

Table of Contents

[Overall Status 3](#_Toc121255712)

[Approach 3](#_Toc121255713)

[Analysis 4](#_Toc121255714)

[File Descriptions 5](#_Toc121255715)

[Division of Labour 5](#_Toc121255716)

# **Overall Status**

**For 3 mapper 1 reducer configuration**:

I have 3 mappers : ActorMapper, TitleMapper, DirectorMapper corresponding to each dataset file and 1 reducer. The input files go to corresponding mappers and the output is the join of all 3 tables joined on titleID. This output is sent to the reducer where the output is manipulated using tags that were added in mappers. In reducer, records are filtered out based on actorID and directorID. Along with that titleID is also matched for each output. Final output consists of all actors who directed the particular title.

**For 3 mapper 2 reducer configuration**:

For this configuration, I have added a custom partitioner and set the number of reduce tasks to zero. The logic remains same as above.

3 mapper and 1 reducer is implemented successfully whereas 3 mapper and 2 reducers is incomplete.

# **Approach**

Mapper code:

Each dataset is sent to the respective mapper. Once data arrives to the mapper function,records of tables have been converted to tokens. While iterating over the tokens,key and value records are identified.

TitleMapper: key: title id, value: titleid, title type, original name and start year.(tag used: titleVal)

In this mapper, I have also filtered data based on the parameters given ie.. the converted the start year to integer and checked that the titles lie between 1193 and 2003. Similarly, only those records have been put to output which had title type as movie.

ActorMapper: key: title id, value: title id, actor id, actor name (tag used: actorVal)

In this mapper, when the data is received it is manipulated to create key and value record.

DirectorMapper: key: title id, value: title id, director id (tag used: directorVal)

In this mapper, same logic is used to get key and value pairs.

Once the output from all mappers are received, it can be inferred that it is out of the type:

<key:title id> <value: actorVal<>,titleVal<>,directorVal<>>

Reducer: once the above output is received from all mappers, a for loop is used to iterate over the values. Each value from the value list is taken and manipulated accordingly. I have created 3 array list to store the data from respective tables. Once the data is segregated, I have used nested for loops to check that for each actor id which are the records with same director id and encapsulated by an outer for loop to match the title id. The output is written in the form <title id as key><actor id, actor name, director id, title name, title type, start year>.

# **Analysis**

Below are the screenshots of time taken by the map reduce job without partition and with partition

1. Without partition

Text

Description automatically generated

1. With partition

Text

Description automatically generated

As it can be inferred with the screenshots, the time taken with the default partition is greater than when using number of reducer jobs as 2. This happens because in case of second case, I have a created a custom partitioner (MyPartitionClass),this function divides the data based on the number of reducer tasks used. In this case , 2 were used so the output from the mappers is divided between two reducing tasks. Thus due to parallel computation both reducer jobs run simultaneously and thus computer faster.

# **File Descriptions**

Following files have been used for the development of project:

1. ActorsDirectorsMR.java : This file contains the code for all 3 mappers, reducer and partitioner. It also contains the configuration for map reduce job.

# **Division of Labour**

Completion of the project can be divided into five sub tasks:

1. Understanding the map reduce programming paradigm
2. Understanding the working of Hadoop
3. Identifying how to join data to get the desired result
4. Writing code for mappers and reducer
5. Testing and debugging

Work done by Sahiba Khan:

* Understanding the map reduce by going through internet sources and asking questions to TA and the professor.
* Installing Hadoop on local system. Resolving issues while installing
* Understanding the flow of Hadoop
* Make myself similar with hdfs commands, Hadoop commands.
* Understanding how map reduce job works
* Started with implementing mapper code, first implemented title mapper. It took sometime as I had to go back and forth to see the results. Once done same was implemented for other mappers.
* Next step was implementing code for reducer. This was the trickiest part. It took a while in understanding how reducer actually uses value list from key value pairs. Also string manipulation in reducer was a complex task.
* Implemented code for partitioner
* Ran the code and checked results.

**M/R configuration details for multiple inputs and other details:**

* For manipulation of value list pair, used arraylist from utils package
* For handling multiple inputs used , MultipleInputs.addInputPath
* Added import org.apache.hadoop.mapreduce.Partitioner for adding custom partitioner