CSE 5331- DBMS Models and Implementation

Project 3: Map/Reduce Programming Exercise and Analysis

Team no.: 30
Group members:
1. Pratik Antoni Patekar (1001937948)
2. Anurag Reddy Pingili (1001863623)

Table of contents

1. Overall status:
We have successfully implemented the Map/reduce code for two configurations, i.e. 3 Mappers/1 Reducer and 3 Mappers/ 2 Reducers. For both configurations, we have created two main functions in the “imdb” class with one reducer function and 3 different Mapper functions (one for each input data file). The difference between both configurations lies in the number of reducers defined in the main function.

2. Approach:
2.1. Mapper Helper classes: We have defined the following four classes to read the required inputs from the files and write the key value pairs output from Mapper to reducer.

1. **class Actor implements Writable:** This class is used to store the values from the imdb00-title-actors dataset as an Actor class object. We defined variables to store the titleID, actorID and actor name values.
2. **class Director implements Writable:** This class is used to store the values from the title-crew dataset as a Director class object. We defined variables to store the titleID, directorID and writerID values.
3. **class Movie implements Writable:** This class is used to store the values from the title-basics dataset as a Movie class object. We defined variables to store the titleID, movie type, primary title, original title, is adult, start year, end year, runtime minutes, and genres values.
4. **class MAD implements Writable:** This class is used to store the values of all the above 3 object types so that we can combine the output from all three mappers and give it as an output to the reducer. We defined variables to store the Actor, Director and Movie object values. We also added a flag variable to identify which values belongs to which class object type. This flag enabled us to differentiate the input received by the reducer and compare the values from different mappers to find a matching actorID and directorID.

2.2. Working of Mapper and reducer code: Following is the detailed explanation of the each Mapper function and reducer function:

1. **Actor_Mapper:** This mapper takes the input from the imdb00-title-actors dataset. It stores the values of titleID, actorID and actorName in 3 variables. We also declared the flag value as “1” in the mapper. We have added some data validation code to ignore the first line of the dataset and then created a variable of the class type Actor and stored the 3 values in the variable. We gave the mapper’s key output as the titleID and the output values as a MAD class type with the actor variable and flag as parameters.

2. **Director_Mapper:** This mapper takes the input from the title-crew dataset. It stores the values of titleID, directorID and writerID in 3 variables, the variable for the directorID being a string array which we split using the delimiter “,”. We also declared the flag value as 2 in the mapper. We used a loop to iterate over the length of the director id string array and then in the loop we have added some data validation code to ignore the first line of the dataset. We then created a variable of the class type Director and stored the 3

values in the variable. We gave the mapper's key output as the titleID and the output values as a MAD class type with the director variable and flag as parameters.

3. **Movie_Mapper:** This mapper takes the input from the title-basics dataset. It stores the values of titleID, movie type, primary title, original title, is adult, start year, end year, runtime minutes, and genres in 9 variables. We also declared the flag value as 3 in the mapper. We checked if the year was a null value ("\N") or a header value. If it was, then we set the year as String "0". We converted the value of the year into an integer. We wrote a condition to ignore the first line of the dataset and then created a variable of the class type Movie and stored the values of titleID, movie type, primary title, start year and genres in the variable. We gave the mapper's key output as the titleID and the output values as a MAD class type with the movie variable and flag as parameters.

4. **Reducer:** We gave the MAD class as the reducer's input and text as the reducer's output. We declared 3 vectors of type Actor, Director and Movie for storing the value of each of the class objects. In reduce function, we iterated over the values of MAD class objects and added the values to each of the 3 vectors based on the value of the flag. After all the values are populated in the vectors, we iterated over all the three vectors and wrote a condition to check for our team parameters inside which we check if the actor id and the director are the same. If they were, we wrote out the values of title id as the key and actor name, movie type, year, movie name as the values.

2.3. How and why the number of mappers and reducers is chosen?

The number of mappers we initially chose was three due to the simple idea of having one mapper for every input file which worked well, so we stuck with it. We tried both the configurations i.e, one reducer and two reducers. The code with two reducers was comparatively faster in execution but only slightly due to the extra time taken for file creation, network transfer to both the reducers and parsing time. We did not keep increasing the number of reducers to make the execution time faster because that would increase the network overhead and the overall time take would be approximately the same.

2.4. Analysis of different M/R configurations:

Generally, for a given code, when we increase the number of reducers, the execution time should decrease because the reducer output is processed parallely by multiple reducers so it decreases the amount of data that each reducer has to handle.

Number of Mappers	Number of Reducers	Runtime(ms)
3	1	325540
3	2	343894
3	3	345682
3	4	335659
6	1	264049
6	2	371427
6	3	327401
6	4	370500

As we can see from the above data, the execution time doesn't directly decrease when we increase the number of reducers. This is because there are a few other factors that effect the time taken for execution. The output produced by the mappers need to be sent to all the reducers that are spread across the network so when there are more reducers, the network transfer time and parsing time increases. As the number of reducers increases, the total number of Input-Output operations that need to be performed also increase. Each reducer has to also be started up and stopped before and after the execution of the reducer code. All of these factors increase the execution time due to which the decrease isn't directly proportional to the number of reducers.

3. File Descriptions:

One folder Project 3 inside which are two folder named 'one' and 'two' for the two configurations i.e, one and two reducers respectively. Inside the folders are the build files, run files, out files and output folder for distributed and local runs.

4. Division of Labor:

Group member name	**Functions implemented**	**Hours spent on project on weekdays**	**Hours spent on project on weekends**
Pratik Antoni Patekar (1001937948)	All the functions were implemented together	Utmost 5 hours	5 to 6 hours
Anurag Reddy Pingili(1001863623)		Utmost 5 hours	5 to 6 hours

5. SQL query and output corresponding to given join problem statement:

Query:

```
select distinct crew.tconst, actors.primaryname, title_basics.titletype, title_basics.primarytitle,
title_basics.genres, title_basics.startyear, crew.directors, actors.nconst
from imdb00.title_crew crew, imdb00.title_basics title_basics,
(select principal.tconst as tconst, principal.nconst as nconst, name_basics.primaryname as
primaryname
from imdb00.title_principals principal, imdb00.name_basics name_basics
where principal.nconst = name_basics.nconst) actors
where crew.tconst = title_basics.tconst
and title_basics.tconst = actors.tconst
and crew.directors like '%' || actors.nconst || '%'
and lower(title_basics.titletype)  = 'movie'
and title_basics.startyear between '1963' and '1973';
```