DBMS Models and implementation
Instructor: Sharma Chakravarthy
Project 3: Map/Reduce and Query Processing

Made available on:	**10/24/2019**
Submit by:	**12/3/2019 (11:55 PM)**
Demo on:	**12/4/2019 (Choice of Slots will be given)**
Submit to:	**Canvas (1 zipped folder containing all the files/sub-folders)**
Weight:	**15% of total**
Total Points:	**100**

One of the advantages of cloud computing is its ability to deal with very large data sets and still have a reasonable response time. Typically, the map/reduce paradigm is used for these types of problems in contrast to the RDBMS approach for storing, managing, and manipulating this data. An immediate analysis of a large data set does not require designing a schema and loading the data set into an RDBMS. Hadoop is a widely used open source map/reduce platform. Hadoop Map/Reduce is a software framework for writing applications which process vast amounts of data in parallel on large clusters. In this project, you will use the IMDB (International Movies) dataset and develop programs to get interesting insights into the dataset using Hadoop map/reduce paradigm. Please use the following links for a better understanding of Hadoop and Map/Reduce (https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)

We have the entire IMDb data set installed as a Relational database on Oracle (on UTA's omega computer). It is a large database with Millions of rows as can be seen from the schema given at the end of this description. The IMDb dataset is publicly available and stores world-wide information about movies, TV episodes, actor, directors, ratings and genres of the movies, etc. This allows us to write SQL queries as well and see whether we can compute the same things as we do on Map/Reduce and also understand query plans using the EXPLAIN statement of Oracle. We can also drill down further on the results of Hadoop analysis.

1. Installation: There are two options

A. **RECOMMENDED - Hadoop Single Node Cluster Setup (Hadoop 2.9.1):** We advise you to use a Linux installation such as Ubuntu 16.04 on your system in order to breeze through the steps of Hadoop cluster set up. You can use a virtual machine for this purpose. The steps to install a *single node cluster in the pseudo distributed mode* are given here - https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html . Some important points that must be considered while setting up the cluster are:

- Preferred Mirror site for download: http://mirrors.ibiblio.org/apache/hadoop/common/hadoop-2.9.1/
- After completing the prerequisites and running the command ($ bin/hadoop), jump to https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation, and follow the

steps for Configuration, Setup passphraseless ssh, Execution (Only steps 1 – 4) and YARN on single node (Steps 1-3)

- The cluster will start up by running the commands,

  $ sbin/start-dfs.sh

  $ sbin/start-yarn.sh

  which you must have done while following the above steps.
- To check if the Namenode, Datanode, Secondary Namenode, ResourceManager, NodeManager are running as separate processes, run the command

  $ jps
- Once the cluster is up, run the most basic code, WordCount (provided), that counts the number of occurrences of each word in the input files. You can also extend it to compute frequency as we have discussed in the class for familiarizing the map/reduce paradigm.

  - Download the WordCount.java code from blackboard into the updated Hadoop folder
  - Make a directory (example, inputFiles) on the HDFS (Hadoop Distributed File System) to store the input files

    $ **bin/hdfs dfs -mkdir** /inputFiles
  - Copy a file from the local file system to the HDFS using the command

    $ **bin/hdfs dfs -put** <path_of_file_on_local_system> /inputFiles
  - To execute the code, follow the steps given here, https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Example:_WordCount_v1.0

- To stop the cluster, run the commands

  $ sbin/stop-dfs.sh

  $ sbin/stop-yarn.sh
- Some useful shell commands for the HDFS can be found here, https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

B. **Cloud Services (self-exploration required):** The second option is to use Amazon Elastic Map/Reduce (or equivalent). Amazon EMR (Elastic Map/Reduce) is a web service provided by Amazon that uses Hadoop and distributes large datasets and processes them into multiple EC2 instances. You have to sign up for AWS. Please make sure you read carefully your AWS agreements/contracts/free use. You may have sufficient free services to complete the projects, but you should monitor and understand your use. Don't leave processes running when not necessary. Note that if you exceed your monthly quota, you will get charged. Also, a credit/debit card is necessary for signing up for this service. You may use other cloud environments like IBM cloud, Google cloud or Microsoft Azure. **For these, we will not be able to provide assistance.** There should be online help available for different cloud providers regarding how to set up a cluster and run the first basic map/reduce code.

Note that if you use Amazon/ECS, you can use multiple mappers and reducers and will be able to better understand and appreciate distributed aspect of map/reduce and how the response time changes (should decrease) with more resources. You may not be able to do this on your laptop installation of Hadoop or on your desktop.

2. **IMDB Dataset (Download Link: http://itlab.uta.edu/downloads/cse5331-datasets/IMDB_TITLES.zip)**

IMDB is a dataset containing information about movies (international) and TV episodes from their beginnings. The information includes movie titles, directors, actors, genre, and year produced/started. Some rating information is also included. The same is true for TV episodes and includes number of seasons in terms of start and end years as well as episodes in each season. It is quite large and should not require additional information to understand the data set. The data of this database that you will use is given below.

IMDB_TITLES.txt: This dataset contains the information about the various **imdb titles** (movies, tv episodes, documentaries etc.), produced across the world. Each field in this dataset is *separated by a semi-colon*. A random sample from this file has been shown below

tt0000091;short;The House of the Devil;1896;Horror,Short
tt0468569;movie;The Dark Knight;2008;Action,Crime,Thriller
tt0088610;tvSeries;Small Wonder;1985;Comedy,Family,Sci-Fi

The description of the various fields has been given below

Field Number	Field Name	Field Description (fields are separated by ";", within a field (e.g., genre) items are separated by ","
1	TITLE ID	The 9-digit unique IMDB title identifier attached to every entry, example: *tt0000091*
2	TITLE TYPE	Every IMDB title in the file is categorized as one of the following TITLETYPEs • *tvSpecial* • *tvMovie* • *tvShort* • *short* • *tvEpisode* • *videogame* • *movie* • *tvSeries* • *tvMiniSeries* • *video*
3	TITLE NAME	The name of the IMDB Title, example: *The Dark Knight*
4	YEAR	The year of release, example: *2008*
5	GENRE LIST	Multiple genres can be attached to a particular title, and they are separated by **commas,** example: *Action,Crime,Thriller*

3. **Project 3 Problem Specification:** You need to compute the following for the given data using the map/reduce paradigm. Try to compare and understand how you would do it using RDBMS if these files were stored as relations. This may help you understand when map/reduce is meaningful and when to use a RDBMS.

   [Task 1] Write a Map/Reduce program to compute the *number of movies* with *different genre combinations* for *pre-defined periods*.

   i) 3 disjoint 5-year periods are being explored as a part of this project – [2001-2005], [2006-2010], and [2011-2015].

   ii) Genre Combinations to be explored
   - Comedy, Romance
   - Action, Thriller
   - Adventure, Sci-Fi

   Example M/R code output for the above

   [2001-2005],Comedy;Romance,<numberOfMovies>
   [2001-2005],Action;Thriller,<numberOfMovies>

   iii) Using the generated result, analyze and plot 3 interesting inferences. *For example, you can plot a histogram of each genre combination for different periods and perform a trend analysis. You can come up with other types of analysis, such as the genre combination that is declining across periods, or genre combination that are going strong across periods.* You can use spreadsheet to post-process the results obtained from the Map/Reduce program to perform the analysis. You can plot histograms, line graphs, scatter plots etc., in order to justify your inferences. Think out of the box!

   You need to design and develop a map program (including a combiner if needed) and a reduce program to solve the above problems. The most important aspects of this design will be to identify the <key, value> pairs to be output by the mapper and computations in the reducer to produce the desired final output.

   [TASK 2] Use the imdb database on omega whose s**chema details are given at the end of this description to** write following SQL queries. **Be careful and aware that this is NOT a toy database as it has Millions of rows in each table and hence your queries can produce tens of thousands of rows as output! Hence, you need to think before posing a query.**

   iv) For ***one of the 3 periods***, write SQL queries to identify **top 5** and **bottom 5 movies** in the above ***genre groups***, based on **movie *rating***. Output the ***movie names, ratings***, and optionally other details, such as ***actor names***.

   v) Using the EXPLAIN statement of Oracle, generate the query plans used in the queries in (iv) and discuss how these queries are evaluated.
   Refer: https://docs.oracle.com/cd/B19306_01/server.102/b14211/ex_plan.htm#i16971

4. **Project Report:** Please include (at least) the following sections in a **REPORT.{txt, pdf, doc}** file that you will turn in with your code:

   i. **Overall Status**
   Give a *brief* overview of how you implemented the major components. If you were unable to finish any portion of the project, please give details about what is completed and your understanding of what is not. (This information is useful when determining partial credit.)

   ii. **Analysis Results:**
   - [TASK 1 – M/R] Explain all the results and related inferences (with graphs if needed) that were drawn.
   - [TASK 2 - SQL] Give the SQL query and NEATLY FORMATTED output of each query along with the query plan. All English queries, SQL equivalents, and their output should be in one output file (named Fall19_<course_num>_<Team<no>.sql)

     For this, you should use the comment capability (--) in SQL to write English queries and use set echo on and spool <filename> and spool off for getting everything in one file. I will briefly go over this in the class. The output should be similar to the ones shown at the end.

   iii. **File Descriptions**
   List the files you have created and *briefly* explain their major functions and/or data structures.

   iv. **Division of Labor**
   Describe how you divided the work, i.e. which group member did what. Please also include how much time each of you spent on this project. (This has no impact on your grade whatsoever; we will only use this as feedback in planning future projects -- so be honest!)

   v. **Logical errors and how you handled them**:
   List at least 3 logical errors you encountered during the implementation of the project. Pick those that challenged you. This will provide us some insights into how we can improve the description and forewarn students for future assignments.

5. **What to submit:**

   - After you are satisfied that your code does exactly what the project requires, you may turn it in for grading. Please submit your project report with your project.
   - You will turn in one zipped file containing you're
     a. **Source code (M/R code, English queries, SQL queries, their output)**
     b. **Outputs from the M/R code**
     c. **Report that includes analysis results for both (spreadsheets etc.) as specified above**
   - All of the above files should be placed in a single zipped folder named as - '**proj3_team_<*TEAM_NO*>**'. **Only one zipped folder should be uploaded using canvas.**
   - You can submit your zip file at most 3 times. The latest one (based on timestamp) will be used for grading. So, be careful in what you turn in and when!

- **Only one person per group should turn in the zip file!**
- 5 days after the due date, the submission will not make sense as there is a penalty of 20% per day (no partial penalties within a day!)

6. Coding style:

Be sure to observe the following standard Java naming conventions and style. These will be used across all projects for this course; hence it is necessary that you understand and follow them correctly. You can look this up on the web. Remember the following:

i. Class names begin with an upper-case letter, as do any subsequent words in the class name.
ii. Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter.
iii. Class, instance and local variables begin with a lower-case letter, and any subsequent words in the name of that variable begin with an upper-case letter.
iv. No hardwiring of constants. Constants should be declared using all upper case identifiers with _ as separators.
v. All user prompts (if any) must be clear and understandable
vi. Give meaningful names for classes, methods, and variables even if they seem to be long. The point is that the names should be easy to understand for a new person looking at your code
vii. Your program is properly indented to make it understandable. Proper matching of if … then … else and other control structures is important and should be easily understandable
viii. Do not put multiple statements in a single line

In addition, ensure that your code is properly documented in terms of comments and other forms of documentation for generating meaningful Javadoc.

7. Grading scheme:

*The **TASK 1 (Map Reduce)** will be graded using the following scheme:*

a.	Correctness of the Map Code	**15**
b.	Correctness of the Reduce Code	**15**
c.	Correctness of the Output	**15**
d.	Analyzing the results obtained ***3 Analysis Results with graphs, plots etc., wherever necessary***	**15**

*The **TASK 2 (SQL Queries)** will be graded using the following scheme:*

e.	Correctness of the SQL Queries and Output	**20**
f.	Query Plan generation using EXPLAIN statement	**10**
g.	Explanation/analysis of query plans	**5**
	Overall Completeness of Report ***Status, File Descriptions, Division of Labor, Logical Errors***	**5**
	TOTAL	**100**

8. Mandatory Project 3 Demonstrations: Mandatory Team-wise demos will be scheduled between **December 3 and December 5, 2019**. A sign-up sheet for the demo Schedule will be provided. If you finish early, you are welcome to demonstrate early by sending the TA an email.

Accessing Oracle on Omega

All of you have access to Omega using the your Netid and password. You should also have access to Oracle whose password is pre-defined for you. In case you do not have access to Oracle, please check with OIT to get access.

Logging into Omega and Oracle

- **i)** Log into Omega using Putty
- **ii)** Type sqlplus {username{/password}}
- **iii)** Prompts (with SQL>) for username and password if not given in the above statement
- **iv)** You can now create tables, insert tuples, drop tables, create views, and run SQL queries.
- **v)** You can also write embedded sql programs, stored procedures, and connect to the database by jdbc or other mechanisms
- **vi)** You have access to the IMDb database created in the database sharmac. You can only access it but not modify.

Some useful commands:

- **i)** SQL> Select *

  From cat; //note the semicolon

  Lists all the tables and views defined by the current user.
- **ii)** SQL>Describe sharmac.title_basics // no semicolon for describe statement

  //you have to prefix all table names with sharmac. As it is in my database!

  Describes the details (attributes and types) of the title_basics table in the database sharmac
- **iii)** help command

  Command is the name for which you want to get help!

  E.g., SQL>help column
- **iv)** You can either enter sql commands at the prompt or put it in a file and submit it.
- **v)** If you put it in a file, use the command

  SQL>@<filename> or start <filename> to execute statements in the file.

  If the file name has .sql extension, you need not specify the extension

  Since you are experimenting, please use a file that you can change and try again!
- **vi)** SQL> Set echo on

  Sets echo on to echo all the input; especially important if you are submitting a file
- **vii)** SQL> Spool <filename>
- **viii)** SQL> spool off

  Is useful to redirect your output to a file for analysis. Created in the current directory.

  You need to set spool off to see its contents. Subsequent runs will append to the same file!

IMDB Database on Omega

We have created a large database and populated it with Millions of rows of International Movies and TV episodes information. It is known as the IMDb database by the community (publicly available data set, but not as a relational DBMS) and used by researchers in databases and other fields. The details of the tables are given below. This has all movie and TV episode information from the beginning (1925) until 2018 for US and international movies and TV episodes. You can query this database for looking up certain information of interest, finding aggregate and statistical information that you are interested in, and OLAP analysis queries as well to the extent possible using SQL.

The IMDb database includes the following information: movie title, year produced, genres a movie belongs to, actors, writers, directors, runtime, adult or non-adult classification, reviews in terms of votes on the movie, average rating, region, language etc. Similarly for TV series.

The purpose of setting up this database is to provide you with an understanding of the differences between a toy DBMS and large real-world DBMS, in terms of the kinds of queries you can ask, the response time, and appreciate the technology behind a DBMS (query optimization, concurrency control, simple relational abstraction, easy-to-use, non-procedural query language etc.)

As many fields contain strings with some delimiter, you need to include the LIKE operator with % and _ for picking out the correct string of interest (can also use string matching). For example, genres can be matched using LIKE 'Comedy' or LIKE 'Drama'. Note the first letter is capitalized. The other genres present are Horror, Short, Thriller, Sci-Fi, Music, Musical, to name a few. For years, use LIKE '200%' to get values in the range 2000 to 2009. Similarly for others. Some populated field values have a \N as their value. So it is useful to have NOT LIKE '\N' to exclude those.

The following tables are populated in the Oracle database on Omega (omega.uta.edu)

Total Number of Tables: 9
Maximum Number of rows in a table: 27 million rows
Maximum Number of attributes in a table: 9; they are self-explanatory.

1. TITLE_BASICS table

```
SQL> describe TITLE_BASICS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 TCONST                                    NOT NULL VARCHAR2(10)
 TITLETYPE                                 NVARCHAR2(500)
```

```
PRIMARYTITLE                    NVARCHAR2(950)
ORIGINALTITLE                   NVARCHAR2(950)
ISADULT                         NUMBER(1)
STARTYEAR                       NUMBER(4)
ENDYEAR                         NUMBER(4)
RUNTIMEMINUTES                  NUMBER(10)
GENRES                          NVARCHAR2(350)

SQL> select count(*) from TITLE_BASICS;

  COUNT(*)
----------
```

Total number of row: 4809386 (4.8 million rows)

****

2. TITLE_CREW_WRITER table

```
SQL> describe TITLE_CREW_WRITER
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 TCONST                          NOT NULL     VARCHAR2(10)
 WRITERS                         NOT NULL     VARCHAR2(10)

 SQL> select count(*) from TITLE_CREW_WRITER;

  COUNT(*)
----------
```

Total number of rows: 5297540 (5.2 million rows)

3. TITLE_CREW_DIR table

```
SQL> describe TITLE_CREW_DIR
 Name                                      Null?          Type
 ----------------------------------------- -------- ----------------------------

 TCONST                          NOT NULL     VARCHAR2(10)
 DIRECTORS                       NOT NULL     VARCHAR2(10)

 SQL> select count(*) from TITLE_CREW_DIR;

  COUNT(*)
----------
```

Total number of rows: 3408484 (3.4 million rows)

****

4. TITLE_EPISODE table

```
SQL> describe TITLE_EPISODE
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 TCONST                                    NOT NULL VARCHAR2(10)
 PARENTTCONST                              NOT NULL VARCHAR2(10)
 SEASONNUMBER                                       NUMBER(9)
 EPISODENUMBER                                      NUMBER(9)

 SQL> select count(*) from TITLE_EPISODE;

  COUNT(*)
----------
   Total number of rows:3206322 (3.2 million rows)
```

****

5. TITLE_PRINCIPALS table

```
SQL> describe TITLE_PRINCIPALS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 TCONST                                    NOT NULL VARCHAR2(10)
 ORDERING                                           NUMBER(4)
 NCONST                                    NOT NULL VARCHAR2(10)
 CATEGORY                                           VARCHAR2(550)
 JOB                                                VARCHAR2(500)
 CHARACTERS                                         NVARCHAR2(800)

 SQL> select count(*) from TITLE_PRINCIPALS;

  COUNT(*)
----------
 Total number of row: 27054380 ( 27 million rows)
```

****

6. TITLE_RATINGS tables

```
SQL> describe TITLE_RATINGS
 Name                                      Null?    Type
```

```
 ----------------------------------------- -------- ----------------------------

 TCONST                                    NOT NULL VARCHAR2(10)
 AVERAGERATING                             NOT NULL NUMBER(5,2)
 NUMVOTES                                  NOT NULL NUMBER(15)

 SQL> select count(*) from TITLE_RATINGS;

  COUNT(*)
----------
 Total number of rows:  805011 ( 0.8 million rows)
```

7. TITLE_AKAS table

```
SQL> describe TITLE_AKAS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 TITLEID                                   NOT NULL VARCHAR2(10)
 ORDERING                                           NUMBER(10)
 TITLE                                              NVARCHAR2(950)
 REGION                                             NVARCHAR2(550)
 LANGUAGE                                           NVARCHAR2(550)
 TYPES                                              NVARCHAR2(550)
 ATTRIBUTES                                         NVARCHAR2(500)
 ISORIGINALTITLE                                    NUMBER(2)

 SQL> select count(*) from TITLE_AKAS;

  COUNT(*)
----------
   3563547 (3.5 million rows)
**********************************************************************
```

8. NAME_TITLE_MAPPING table

```
SQL> describe NAME_TITLE_MAPPING
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 NCONST                                    NOT NULL VARCHAR2(10)
 TCONST                                    NOT NULL VARCHAR2(10)

SQL> select count(*) from NAME_TITLE_MAPPING;
```

```
  COUNT(*)
----------
  14144524 (14 million rows)
**********************************************************************
```

9. NAME_BASICS table

```
SQL> describe NAME_BASICS
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------

 NCONST                                    NOT NULL VARCHAR2(10)
 PRIMARYNAME                               NOT NULL NVARCHAR2(950)
 BIRTHYEAR                                          NUMBER(4)
 DEATHYEAR                                          NUMBER(4)
 PRIMARYPROFESSION                                  VARCHAR2(900)

SQL> select count(*) from NAME_BASICS;

  COUNT(*)
----------
  8424762 (8.4 million rows)
```