

INFSCI 2725 Data Analytics

Assignment 1 - Distributed Storage and Processing of Data

Date: 9/29/2015

Student name: Ting LI, Hongyuan CUI, Zhe ZHANG

Tutor name: Marek Druzdzal

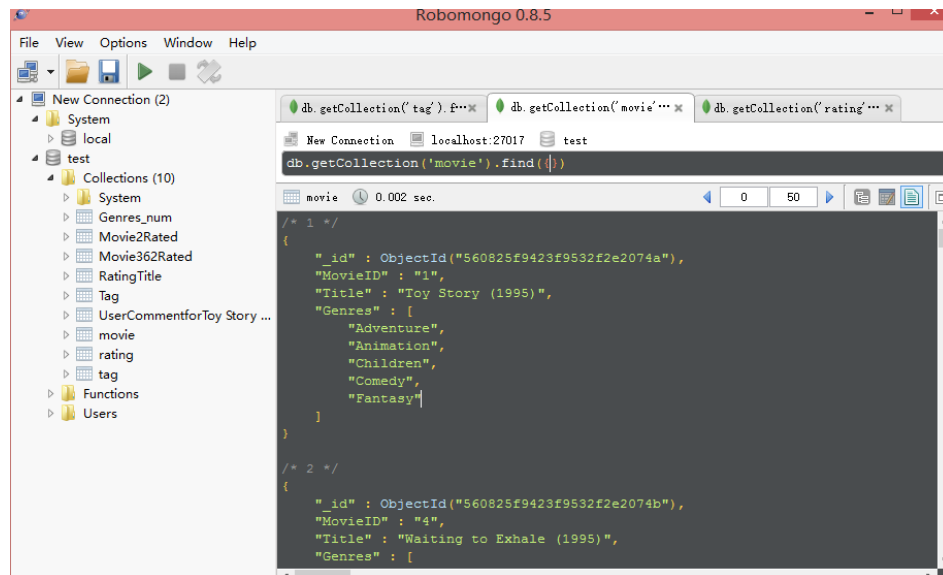
Introduction

By examining the data of this assignment and the queries we need to carry out, we decided to choose MongoDB as the database to process the data. During the implementation of this assignment, we have used a visualization tool to optimize the interaction with the data set. We created a data store, transformed the data and made queries with programs in Java. The detailed method of this assignment is presented and explained as below.

Main Content

1. Installation of MongoDB and Visualization of data input;

Following the MongoDB manual, the MongoDB was successfully installed and running environment was set up as well accordingly. Visualization software, called Robomongo, was then introduced and put into use in order to have a better interaction during the data analytics process. The flowing is a picture to show this step,



2. Transform the data into Json format.

Having looked at the synopsis of MongoDB from its website, we noticed that MongoDB can only import content from an Extended JSON, CSV, or TSV format, therefore, transformation

of the given data set is necessary. Following is a snapshot of data transformation for movies.dat, and please refer to the FileTransform program for detailed code.

```

reader = new BufferedReader(new FileReader(file));
while((str=reader.readLine())!=null){
    //Transform movies.dat
    jsonObject = new JSONObject();
    String[] each=str.split("::");
    jsonObject.put("MovieID", each[0]);
    jsonObject.put("Title", each[1]);
    String [] genre = each[2].split("\\|");
    String[] array = new String[genre.length];
    for(int i=0;i<genre.length;i++){

        array[i]=genre[i];

    }
    jsonObject.put("Genres", array);
    Write(jsonObject.toString()+"\n");
}

```

3. Import the transformed data set into MongoDB;

In the command prompt, we use “mongoimport” to import the data set into the test database and split the data set into three aggregations.

The following is the command line for importing the data in the command prompt:

mongoimport --db test --collection movie --drop --file C:\Desktop\Json_file\movie.json

Please refer to the flowing picture for this step,

```

C:\WINDOWS\system32\cmd.exe
/682.0 MB <98.6%>
2015-09-27T13:22:33.805-0400 [#####.1 test.rating 676.5 MB
/682.0 MB <99.2%>
2015-09-27T13:22:36.805-0400 [#####.1 test.rating 681.3 MB
/682.0 MB <99.9%>
2015-09-27T13:22:37.480-0400 imported 10000096 documents

c:\mongodb\bin>mongoimport --db test --collection movie --drop --file C:\Users\David\Desktop\Json_File\Json_File\movie.json
2015-09-27T13:23:05.631-0400 connected to: localhost
2015-09-27T13:23:05.636-0400 dropping: test.movie
2015-09-27T13:23:06.171-0400 imported 10681 documents

c:\mongodb\bin>mongoimport --db test --collection tag --drop --file C:\Users\David\Desktop\Json_File\Json_File>tag.json
2015-09-27T13:23:23.776-0400 connected to: localhost
2015-09-27T13:23:23.782-0400 dropping: test.tag
2015-09-27T13:23:26.765-0400 [#####.1 test.tag 2.3 MB/7
.4 MB <31.3%>
2015-09-27T13:23:29.765-0400 [#####.1 test.tag 3.9 MB/7
.4 MB <52.3%>
2015-09-27T13:23:32.613-0400 imported 95580 documents

c:\mongodb\bin>_

```

4. Implementation of the queries and part of the results;

All program codes for each query are in the attached files (for example, the code for Query 1 is Question1, the rest follows this rule). Explanation of the queries we asked and some results are as follows.

Query1 Write a query that finds average rating of each movie

- At first, because of the demand for the average rating of each movie in the first query, we use accumulators to perform calculations for each group with “\$group”, the code is as follows,

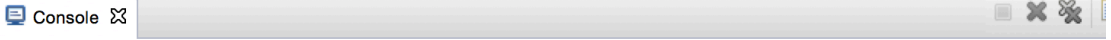
```
AggregateIterable<Document>iterable = ratingcol.aggregate (asList(newDocument("$group",new Document("_id", "$MovieID").append("count", new Document("$avg", "$Rating")))));
```

- Afterwards, we use the following code to retrieve the MovieTitle with MovieID for each movie from movie collection

```
FindIterable<Document> findMovie = movie.find (newDocument ("MovieID", document1.get("_id")));
```

- Finally, we put movie titles and average rating into a new collection, `db.getCollection("AverageRating").insertOne(newDocument("Title",document2.get("Title")).append("AvgRating",document1.getDouble("avgRating")));`

Part of the result is showing below,



```
<terminated> Question1 [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java (Sep 28, 2015, 2:01 PM)
Title: Hamlet (1964) AvgRating: 3.0708661417322833
Title: Funny Girl (1968) AvgRating: 3.852201257861635
Title: Solaris (Solyaris) (1972) AvgRating: 3.8219178082191783
Title: 3 Days of the Condor (a.k.a. Three Days of the Condor) (1975) AvgRating: 3.6885245901639343
Title: Happiness Is in the Field (Bonheur est dans le pré, Le) (1995) AvgRating: 3.857142857142857
Title: Back to the Future (1985) AvgRating: 4.077151335311573
Title: Iron Eagle II (1988) AvgRating: 4.0
Title: Monster in the Closet (1986) AvgRating: 3.9261744966442955
Title: Lucky Number Slevin (2006) AvgRating: 2.7083333333333335
Title: Airport '77 (1977) AvgRating: 3.561525129982669
Title: Big (1988) AvgRating: 3.953846153846154
Title: Anchorman: The Legend of Ron Burgundy (2004) AvgRating: 3.263265306122449
Title: Airplane II: The Sequel (1982) AvgRating: 3.3933333333333335
```

Query2 Write a query that finds users who are similar to a given user (target user), the id of the target user is an input parameter. Users are similar to the target user if they rate the same movies.

- The input of the target user ID is as follows,

```
BufferedReader br = new BufferedReader (new InputStreamReader(System.in));
System.out.println ("Enter your value:");
Final String str = br.readLine();
System.out.println("your value is :"+str);
```

- Then find target user’s UserID document in rating collection, `FindIterable<Document> findUser = rating.find(new Document("UserID", userID));`

- Find similar Users according to MovieID in the rating collection,

```
FindIterable<Document> findMovie = rating.find(new Document ("MovieID",document1.get("MovieID")));
```

- Finally, add similar users into a set and print them

```
userSet.add(document2.get("UserID").toString());
System.out.println(userSet);
```

Part of the result is showing below,

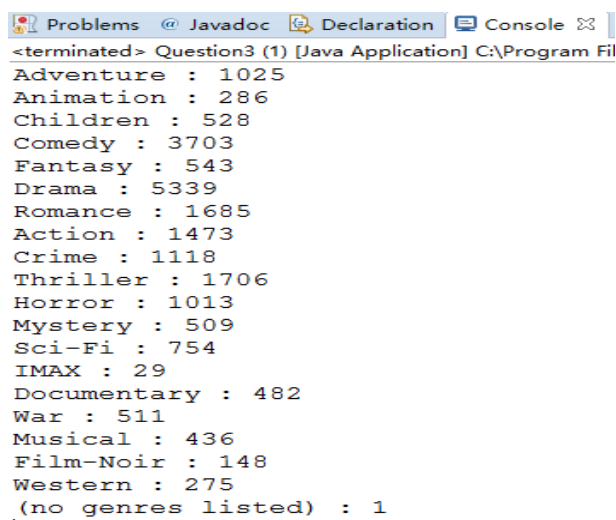
```
[1, 10, 100, 1000, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 101, 1010, 1011, 1012, 1013, :
[1, 10, 100, 1000, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 101, 1010, 1011, 1012, 1013, :
[1, 10, 100, 1000, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 101, 1010, 1011, 1012, 1013, :
```

Query3 Write a query that finds to number of movies in each genre.

- This query can be done in the movie collection. The query asked is as below,

```
DBCollection dbcoll = db2.getCollection("movie");
List list=dbcoll.distinct("Genres");
for( int i=0;I < list.size(); i++)
{
    FindIterable<Document>findGenresIdcol=db.getCollection("movie").find(new
    Document("Genres", list.get(i)));
    findGenresIdcol.forEach(new Block<Document>() {
    @Override
    public void apply(final Document document) {
        num++;}
    }
```

Part of the result is showing below,



```
<terminated> Question3 (1) [Java Application] C:\Program Fil
Adventure : 1025
Animation : 286
Children : 528
Comedy : 3703
Fantasy : 543
Drama : 5339
Romance : 1685
Action : 1473
Crime : 1118
Thriller : 1706
Horror : 1013
Mystery : 509
Sci-Fi : 754
IMAX : 29
Documentary : 482
War : 511
Musical : 436
Film-Noir : 148
Western : 275
(no genres listed) : 1
```

In order to demonstrate that our data storage is working, we produce 3 more queries to test it. They are as follows,

Query4

In this query, we want to list all the tags for one movie,

- Firstly, enter a movie title as follows,

```
System.out.println("You can enter the movie title to see all of its tags");
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
final String title = br.readLine();
System.out.println("You entered "+title);
```

- Find the Title of the movie entered,

```
FindIterable<Document> findMovie = movie.find(new Document("Title", title));
```

- Get the id of those movies this user rated,

```
FindIterable<Document> findComment = tag.find(new Document("MovieID", document1.get("MovieID")));
```

- Finally, get the tags for the target movie,

```
buffer.append("UserID:"+document2.get("UserID")).append(" ");
buffer.append("Tag: "+document2.get("Tag")).append("\n");
System.out.println(buffer.toString());
```

```
db.getCollection("UserCommentfor"+title).insertOne(new
Document("UserID",document2.get("UserID")).append("Tag", document2.get("Tag")));
The result is as below,
```

```
You can enter the movie title to see all of its tags
Toy Story (1995)
You entered Toy Story (1995)
Sep 28, 2015 3:20:36 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SING
Sep 28, 2015 3:20:37 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: No server chosen by ReadPreferenceServerSelector{readPreference=p
Sep 28, 2015 3:20:37 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:1, serverValue:104}] to
Sep 28, 2015 3:20:37 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Monitor thread successfully connected to server with description:
Sep 28, 2015 3:20:37 PM com.mongodb.diagnostics.logging.JULLogger log
INFO: Opened connection [connectionId{localValue:2, serverValue:105}] to
UserID: 1751 Tag: Pixar
UserID: 2030 Tag: Pixar
UserID: 2346 Tag: Pixar
UserID: 2456 Tag: animation
UserID: 2456 Tag: Pixar
UserID: 4697 Tag: animated
UserID: 4697 Tag: fun
UserID: 5002 Tag: toy
UserID: 5002 Tag: toys
UserID: 8043 Tag: pixar
UserID: 8302 Tag: rated-G
UserID: 9292 Tag: Pixar
UserID: 9316 Tag: pixar
UserID: 10555 Tag: National Film Registry
```

Query5

In this query, we want to find out the movie title and corresponding genre one user rated.
The major part of this query is as follow,

- Find the id of those movies the target user rated

```
FindIterable<Document> findUser = rating.find(new Document("UserID", userID));  
FindIterable<Document> findMovie = movie.find(new Document("MovieID", document1.get("MovieID")))
```

- Then write the result into collection and output

```
buffer.append("Title: "+document2.get("Title")).append(" ");  
buffer.append("Genres: "+document2.get("Genres")).append("\n");  
System.out.println(buffer.toString());
```

```
db.getCollection("Movie"+userID+"Rated").insertOne(new Document("Title",document2.get("Title")).append("Genres", document2.get("Genres")));
```

Part of the result is showing below,

```
Enter the userID to see all the movies he/she rated  
15  
You entered 15  
Sep 28, 2015 3:17:35 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: Cluster created with settings {hosts=[localhost:27017], mode=SINGLE, requiredClusterType=  
Sep 28, 2015 3:17:35 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: No server chosen by ReadPreferenceServerSelector{readPreference=primary} from cluster des  
Sep 28, 2015 3:17:35 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: Opened connection [connectionId{localValue:1, serverValue:102}] to localhost:27017  
Sep 28, 2015 3:17:35 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: Monitor thread successfully connected to server with description ServerDescription{address  
Sep 28, 2015 3:17:35 PM com.mongodb.diagnostics.logging.JULLogger log  
INFO: Opened connection [connectionId{localValue:2, serverValue:103}] to localhost:27017  
Title: Dead Man Walking (1995) Genres: [Crime, Drama]  
Title: Chungking Express (Chóngqīng Sēnlín) (1994) Genres: [Drama, Mystery, Romance]  
Title: Target (1995) Genres: [Action, Drama]  
Title: Brothers McMullen, The (1995) Genres: [Comedy]  
Title: Showgirls (1995) Genres: [Drama]  
Title: Before Sunrise (1995) Genres: [Drama, Romance]  
Title: Shawshank Redemption, The (1994) Genres: [Drama]  
Title: To Live (Huozhe) (1994) Genres: [Drama]  
Title: Tommy Boy (1995) Genres: [Comedy]  
Title: Body Snatchers (1993) Genres: [Horror, Sci-Fi, Thriller]  
Title: Second Best (1994) Genres: [Drama]  
Title: War Room, The (1993) Genres: [Documentary]  
Title: Nelly & Monsieur Arnaud (1995) Genres: [Drama]  
Title: Multiplicity (1996) Genres: [Comedy]  
Title: Honey Moon (Honigmond) (1996) Genres: [Comedy]  
Title: Maya Lin: A Strong Clear Vision (1994) Genres: [Documentary]  
Title: Grass Harp, The (1995) Genres: [Comedy, Drama]  
Title: Spirits of the Dead (Histoires extraordinaires) (1968) Genres: [Horror, Mystery]  
Title: Supercop 2 (Project S) (Chao ji ji hua) (1993) Genres: [Action, Comedy, Crime, Thriller]  
Title: Wizard of Oz, The (1939) Genres: [Adventure, Children, Fantasy, Musical]  
Title: Scarlet Letter, The (1926) Genres: [Drama]  
Title: Fly Away Home (1996) Genres: [Adventure, Children]  
Title: E.T. the Extra-Terrestrial (1982) Genres: [Children, Drama, Sci-Fi]  
Title: Escape from New York (1981) Genres: [Action, Adventure, Sci-Fi, Thriller]  
Title: ...
```

Query6

In this query, we want to find out genres with a given movie ID

- Find the documents with a given ID;

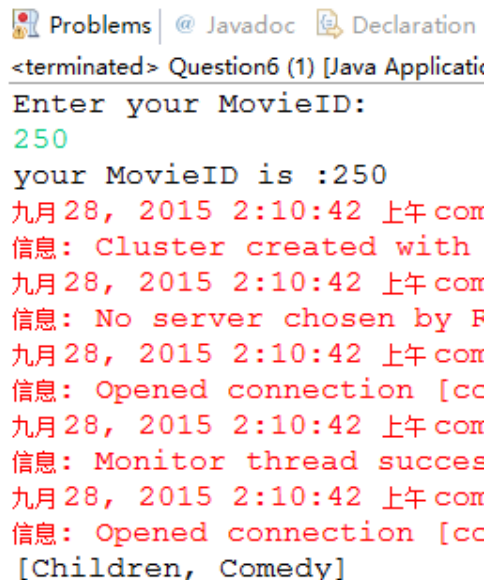
```
FindIterable<Document> findMovieIdcol=db.getCollection("movie").find(new Document("M
```

```
ovieID",str));
```

- Then find Genres of the given MovieID;

```
FindIterable<Document>findMovieIdcol2=db.getCollection("movie").find(newDocument("MovieID",document.get("Genres"))));
```

The result is as follow,



```
Problems | @ Javadoc | Declaration
<terminated> Question6 (1) [Java Applicatio
Enter your MovieID:
250
your MovieID is :250
九月28, 2015 2:10:42 上午 com
信息: Cluster created with
九月28, 2015 2:10:42 上午 com
信息: No server chosen by F
九月28, 2015 2:10:42 上午 com
信息: Opened connection [cc
九月28, 2015 2:10:42 上午 com
信息: Monitor thread succes
九月28, 2015 2:10:42 上午 com
信息: Opened connection [cc
[Children, Comedy]
```

Conclusion and recommendations

This is a very good assignment for students to practice and understand the storage and processing of data. All members in the group have made great effort to accomplish the results.

Ting Li extracted, transformed the data set into Json format, carried out optimization of the programs, and composed Query 4 and 5.

Hongyuan Cui did the major part of the program in this assignment, particularly in Query 1, 2 and 3, and shared his experiences in programming with Java.

Zhe Zhang drafted the report, composed Query 6, and provided assistance for the group members in this assignment.

In addition to the achievement mentioned above, we hope we can get improvement in building data model, fine-tune program code and have better understanding in data storage and processing.