Final project_Yan Lyu

by Yan Lyu

Submission date: 14-Dec-2019 10:03PM (UTC-0500)
Submission ID: 1234654873
File name: FinalProjectFor7250.pdf (10.28M)
Word count: 14947
Character count: 107938

ARTIST DATA ANALYST

Yan Lyu
001493900

TABLE OF CONTENTS

DATASET

This dataset contains social networking, tagging, and music artist listening information from a set of 2K users from Last.fm online music system. The dataset is released in the framework of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec 2011) at the 5th ACM Conference on Recommender Systems (RecSys 2011)

DATASET LINK:

http://files.grouplens.org/datasets/hetrec2011/hetrec2011-lastfm-2k.zip

INTRODUCTION

This is some data about users, artists and tags. In this data set, artists are used to represent songs. The user listens to different artists, the same artist is tagged with different tags by different users, and the same user has a list of friends.
There is a total of 6 files in this data set, which represented as the following:

1. artists.dat (17632)	
This file contains 17632 records and is the artist's detailed information. My main use of this file is to extract the details of the artist, such as name.	
Id, name, url, pictureURL	Artists ' ID, name and url
2. tags.dat (11946) ：	
This file has 11946 records containing tag information. Tags are similar to a favorite list, each user has several favorite lists, and each list has different songs.	
tagID, tagValue	All tag information, including tag id and name
3. user_artists.dat (92834)	
This file has 92834 records, which records the user and all the songs he has listened to, listen times for each song. It can tell us what the user's favorite song is and how many times each song has been listened to.	
userID, artistID, weight	Artists and times for each user
4. user_taggedartists.dat (186479) ：	
This file has 186479 records, including the time when a song was added to a tag list.	
userID,artistID, tagID, day, month, year	The tag and corresponding time (indicated by year, month, and day) marked by each user for each artist.
5. user_taggedartists-timestamps.dat	
This file has 186479 records, including the time when a song was added to a tag list. Time is inferred by timestamp.	
userID, artistID, tagID, timestamp	Tag and corresponding timestamp marked by each user for each artist
6. user_friends.dat (25434 pairs)	
userID, friendID	Correspondence of the user and his friends

PROBLEM STATEMENT

I will do some work on this data set based on users' behaviors. Works include:

1. Analyzing the original data, including analysis of the most popular artists each year through join
2. Making recommendations based on user behavior. Using Mahout to build recommenders, and map reduce to build content-based recommendations
3. Finding common friends of the user
4. Performance comparison between map reduce, hive and pig
5. Using bloom filter, distributed catch, DB map and other technologies to optimize the map reduce process.
6. Using Job Control and threads to achieve parallel computing of multiple jobs and set dependencies for the jobs

WORKS SUMMARY

ALGORITHM 1: FIND THE TOP 10 ARTISTS FOR EACH YEAR.

1. Map Reduce: JobControl (13 jobs), thread, custom key writable, secondary sort, custom partitioner, spliting files
    a. Job 1: Remove duplicated records. Dataset: user_taggedartists.dat
    b. Job 2: Join 2 datasets. A custom key writable object is used, and the results are more readable through secondary sorting. Dataset : user_artist.dat & generated file of job 1
    c. Job 3: Parting file into 10 subfiles.
    d. Job 4 – Job 13: Find the top 10 for each subfile
    e. Connect all jobs through chaining jobs.
2. Hive: Common table expression, Join
3. Pig: Join
4. Performance comparation

ALGORITHM 2: FIND THE TOTAL NUMBER OF ARTISTS EACH YEAR

1. 2 jobs, custom value writable, bloom filter, distributed cache, get sums
2. Job 1: Get the total times a user listened on this platform. Dataset: user_artist.dat
3. Build the bloom filter and store it in the distributed cache
4. Job 2: Load bloom filter from Distributed Cache and filtering users who do not meet the requirements through the bloom filter in mapper and find the total number of songs played on this platform each year using dataset: user_taggedartists.dat.

ALGORITHM 3: RECOMMENDED SONG PLAYLIST

1. Mahout recommender & DB Map
    a. Mahout is used to build the recommender, user-based recommendation
    b. DB Map is used to get the artist name according to artist id
2. Map Reduce: 4 jobs, collaborative filtering, custom value writable
    a. Job 1: Extracting rating information from dataset : user_artists.dat

b. Job 2: Calculate 7 statistical data to be used in the next step
c. Job 3: Calculate Pearson Correlation, Cosine Correlation and Jaccard Correlation
d. Job 4: Recommendations are made based on three correlation coefficients and produce a recommendation result.

ALGORITHM 4: FIND COMMON FRIENDS

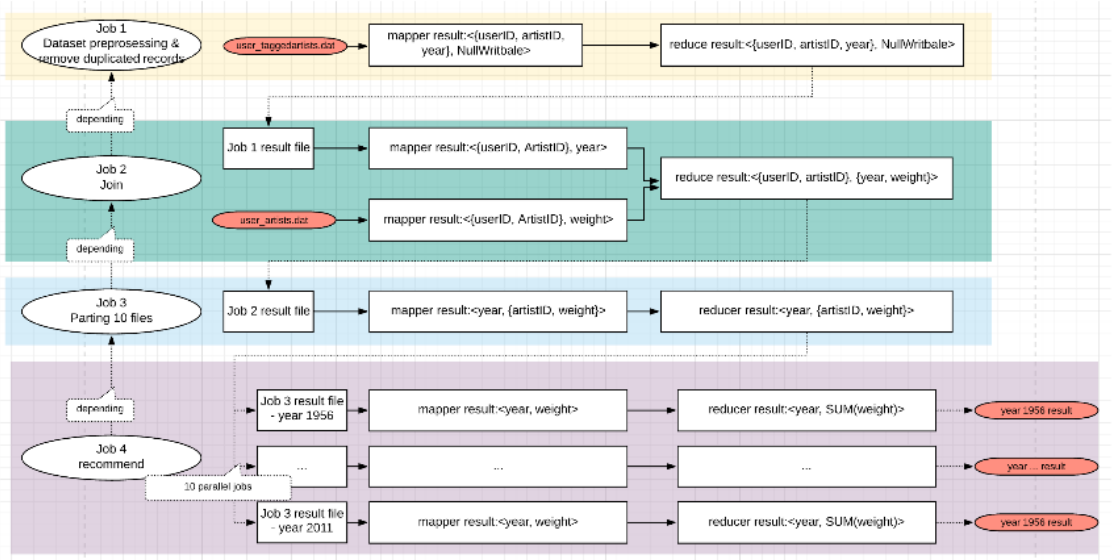
1. Job 1: Convert data format
2. Job 2: Find the common friends of the two users by calculating intersections.

MAP REDUCE OPTIMIZATION CONCLUSION

1. Used DB Map to store the artist.dat file into my disk for getting the artist name. By doing this, I can reduce the number of tables used for join
2. Used Bloom Filter to define the regular user who listened more than 500 song in the platform. By doing this, I don't need to join the user_artist.dat file with other files all the time.
3. Used Hive and Pig, their own optimization engine is very helpful for improving performance.

FIND THE MOST POPULAR SONGS OF THE YEAR & JOBCONTROL & SECONDARY SORTING & PARTING FILE

This algorithm uses a total of 13 jobs to determine the top 10 songs that are played the most each year.
The workflow is:



I used 4 chaining jobs; these 4 jobs are controlled by a job control. Each job depends on the previous job, and the next job can be executed only after the execution of the previous job is completed. In the 4th job, there are 10 parallel jobs. There are no dependencies between them. They act on 10 separate files and can be executed parallel.

WAY 1 : MAP REDUCE

This is part of the annual report of the music system that lets users and artists know what the ten most popular artists are this year.

Dataset

The dataset needed are:

1. user_artists.dat provides information about music being listened to.
2. user_taggedartists.dat provide the year information of songs.

Header of user_taggedartists.dat

userID	artistID	weight

Header of user_taggedartists.dat

userID	artistID	tagID	day	month	year

For the second dataset, only the userID, artistID and year fields are selected. And I found that only selecting the three fields of userID, artistID, and year will create duplicate records, so I must remove duplicate records firstly.
The basis of join is when userID and artistID are equals at the same time, by this way, you can know the number of times a song was heard by each user in a certain year.
Header after join：

userID	artistID	weight	year

JOB1: REMOVE THE DUPLICATED RECORDS

Preprocess user_taggedartists.dat (provide year information). This job includes selecting useful information and removing duplicate records.

MAPPER:

Select the 0th, 1st, and 5th fields of each row, which represent userID, artistID, and year respectively. Combine these three fields into a key and emit a NullWritable value. This is in preparation for removing duplicate records.

REDUCER:

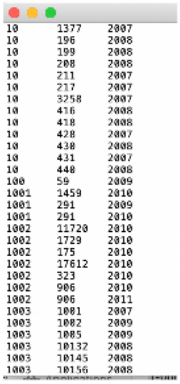
Each Reducer outputs only one record, eliminating duplicate records.

```
@Override
protected void reduce(Text key, Iterable<NullWritable> values,
        Reducer<Text, NullWritable, Text, NullWritable>.Context context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    context.write(key, NullWritable.get());
}
```

RESULT:

This is the result after extracting the userid, artistid, and year and removing duplicate records.

userid	artistid	year



```
10      1377    2007
10      196     2008
10      199     2008
10      208     2008
10      211     2007
10      217     2007
10      3258    2007
10      416     2008
10      418     2008
10      428     2007
10      438     2008
10      431     2007
10      448     2008
100     59      2009
1001    1459    2010
1001    291     2009
1001    291     2010
1002    11720   2010
1002    1729    2010
1002    175     2010
1002    17612   2010
1002    323     2010
1002    906     2010
1002    906     2011
1003    1001    2007
1003    1002    2009
1003    1005    2009
1003    10132   2008
1003    10145   2008
1003    10156   2008
```

JOB2: JOIN TWO DATASETS & SECONDARY SORTING

MAPPER1:

Process year data. Key is {userID, artistID} pair, and value is year. All the values in this dataset are tagged as 'A'

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, UserArtistWritable, Text>.Context context)
        throws IOException, InterruptedException {
    String[] tokens = value.toString().split("\\t");
    int userID = Integer.parseInt(tokens[0]);
    int artistID = Integer.parseInt(tokens[1]);
    outKey = new UserArtistWritable(userID, artistID);
    String year = tokens[2];
    outValue = new Text("A" + year);
    context.write(outKey, outValue);
}
```

MAPPER 2

Process count data. Key is {userID, artistID}, and value is count. All the values in this dataset are tagged as 'B'.

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, UserArtistWritable, Text>.Context context)
        throws IOException, InterruptedException {
    if(key.get() == 0) {
        return;
    }
    String[] tokens = value.toString().split("\\t");
    int userID = Integer.parseInt(tokens[0]);
    int artistID = Integer.parseInt(tokens[1]);
    outKey = new UserArtistWritable(userID, artistID);
    String count = tokens[2];
    outValue = new Text("B" + count);
    context.write(outKey, outValue);
}
```

SECONDARY SORTING: SORTCOMPARATOR & GROUPCOMPARATOR:

A custom Key is used here, and the key is is {userID, artistID} pair. All records are compared according to userID, and then according to artistID, both in ascending order.

```
public class SecondarySortComparator extends WritableComparator{
    public SecondarySortComparator() {
        super(UserArtistWritable.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        UserArtistWritable ckw1 = (UserArtistWritable) a;
        UserArtistWritable ckw2 = (UserArtistWritable) b;
        int result = ckw1.getUserID() - ckw2.getUserID();
        if(result == 0) {
            result = ckw1.getArtistID() - ckw2.getArtistID();
        }
        return result;
    }
}
```

```
public class GroupComparator extends WritableComparator{

    public GroupComparator() {
        super(UserArtistWritable.class, true);
    }

    @Override
    public int compare(WritableComparable a, WritableComparable b) {
        UserArtistWritable ckw1 = (UserArtistWritable) a;
        UserArtistWritable ckw2 = (UserArtistWritable) b;
        int result = ckw1.getUserID()-ckw2.getUserID();
        if(result == 0) {
            result = ckw1.getArtistID()-ckw2.getArtistID();
        }
        return result;
    }
}
```

PARTITIONER:

Partition by artistID. This means that records with the same artistID will be assigned to the same reduce task.

```
public class PartitionerComparator extends Partitioner<UserArtistWritable, Text>{

    @Override
    public int getPartition(UserArtistWritable key, Text value, int num) {
        // TODO Auto-generated method stub
//      return 0;
        return key.getArtistID() % num;
    }

}
```

REDUCER :

The principle of join is inner join, and userID is no longer needed in the subsequent steps. The output of this reducer is: key is year, value is {artistid, count} pair.
Code for joining:

```
@Override
protected void reduce(UserArtistWritable key, Iterable<Text> values,
        Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    // year, count
    List<Integer> yearList = new ArrayList<Integer>();
    List<Integer> countList = new ArrayList<Integer>();
    int year = 0;
    int count = 0;
    for (Text t : values) {
        String value = t.toString();
        if (value.charAt(0) == 'A') {
            year = Integer.parseInt(value.substring(1));
            yearList.add(year);
            min = Math.min(min, year);
            max = Math.max(max, year);
        } else if (value.charAt(0) == 'B') {
            count = Integer.parseInt(value.substring(1));
            countList.add(count);
        }
    }

    for (int y : yearList) {
        String tuple = y + "\t" + key.getArtistID() + "\t";
        for (int c : countList) {
            tuple += c + "";
            context.write(new Text(tuple), NullWritable.get());
        }
    }
}
```

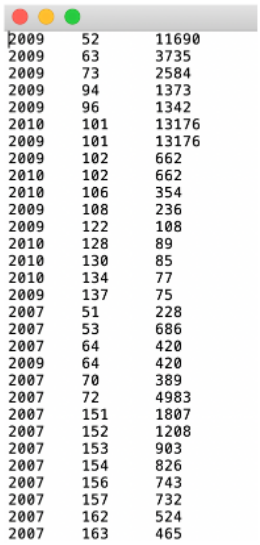
When joining, use a set to record all occurrences of the year. In order to facilitate the next job to divide the data set by year.
Code for record year information:

```
@Override
protected void setup(Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    min = Integer.MAX_VALUE;
    max = Integer.MIN_VALUE;
}

@Override
protected void cleanup(Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    context.getConfiguration().setInt("min.last.access.date.year", min);
    context.getConfiguration().setInt("num", max - min + 1);
    System.out.println("In the join reduceer: " + context.getConfiguration().getInt("min.last.access.date.year", 111));
    System.out.println("In the join num: " + context.getConfiguration().getInt("num", 111));
}
```

RESULT:

year	artistID	weight



```
2009    52      11690
2009    63      3735
2009    73      2584
2009    94      1373
2009    96      1342
2010    101     13176
2009    101     13176
2009    102     662
2010    102     662
2010    106     354
2009    108     236
2009    122     108
2010    128     89
2010    130     85
2010    134     77
2009    137     75
2007    51      228
2007    53      686
2007    64      420
2009    64      420
2007    70      389
2007    72      4983
2007    151     1807
2007    152     1208
2007    153     903
2007    154     826
2007    156     743
2007    157     732
2007    162     524
2007    163     465
```

JOB 3: PARTING TO 10 FILES

Divide the data into 10 sub-files by year. By doing this, we can just consider the data of one year at one time. Which helps us sort out the logic.

MAPPER

The key of the output data is year, and the value is {artistID, weight}

PARTITIONER

According to the statistics of the previous job, we can know that there is a total of 10 years of information in the data set, so the number of reducer tasks is set to 10.
In the partitioner, a reduce task is assigned for each year. But there is a gap between each year, and all of them need to be handled separately.
The years of the data are: 1956, 1957, 1976, 2005, 2006, 2007, 2008, 2009, 2010, 2011. They each correspond to a reducer task.

```java
public class YearPartitioner extends Partitioner<IntWritable, Text> implements Configurable{

    private static final String MIN_LAST_ACCESS_DATE_YEAR = "min.last.access.date.year";
    private Configuration conf = null;
    private int minLastAccessDateYear = 0;

    public Configuration getConf() {
        // TODO Auto-generated method stub
        return conf;
    }

    public void setConf(Configuration conf) {
        // TODO Auto-generated method stub
        this.conf = conf;
        minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
    }

    @Override
    public int getPartition(IntWritable key, Text value, int num) {
        // TODO Auto-generated method stub
//      return 0;
//      return key.get() - minLastAccessDateYear;
        if(key.get() == 1956) {
            return 0;
        }else if(key.get() == 1957) {
            return 1;
        }else if(key.get() == 1979) {
            return 2;
        }else {
            return key.get() - 2005 + 3;
//          return 0;
        }
    }
    public static void setMinLastAccessDateYear(Job job, int minLastAccessDateYear) {
        job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR, minLastAccessDateYear);
    }

}
```

REDUCER

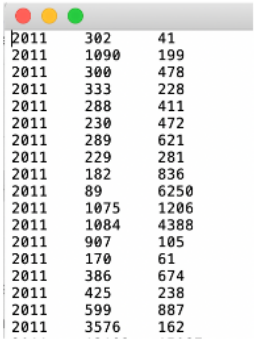
Direct output of the original Key and Values. Because we just split the data based on year and don't want to change the original value of the data. The work of dividing the data has been completed by a custom partitioner.

RESULT

A large file is divided into ten small files by year.

☐	-rw-r--r--	lvyan	supergroup	14 B	Dec 11 22:00	1	128 MB	part-r-00000	🗑
☐	-rw-r--r--	lvyan	supergroup	14 B	Dec 11 22:00	1	128 MB	part-r-00001	🗑
☐	-rw-r--r--	lvyan	supergroup	0 B	Dec 11 22:00	1	128 MB	part-r-00002	🗑
☐	-rw-r--r--	lvyan	supergroup	4.43 KB	Dec 11 22:01	1	128 MB	part-r-00003	🗑
☐	-rw-r--r--	lvyan	supergroup	17.8 KB	Dec 11 22:01	1	128 MB	part-r-00004	🗑
☐	-rw-r--r--	lvyan	supergroup	37.94 KB	Dec 11 22:01	1	128 MB	part-r-00005	🗑
☐	-rw-r--r--	lvyan	supergroup	67.88 KB	Dec 11 22:01	1	128 MB	part-r-00006	🗑
☐	-rw-r--r--	lvyan	supergroup	75 KB	Dec 11 22:01	1	128 MB	part-r-00007	🗑
☐	-rw-r--r--	lvyan	supergroup	90.67 KB	Dec 11 22:01	1	128 MB	part-r-00008	🗑
☐	-rw-r--r--	lvyan	supergroup	27.79 KB	Dec 11 22:01	1	128 MB	part-r-00009	🗑

year	artistID	weight



```
2011    302     41
2011    1090    199
2011    300     478
2011    333     228
2011    288     411
2011    230     472
2011    289     621
2011    229     281
2011    182     836
2011    89      6250
2011    1075    1206
2011    1084    4388
2011    907     105
2011    170     61
2011    386     674
2011    425     238
2011    599     887
2011    3576    162
```

JOB 4: GET THE TOP 10

Each year's data is now processed separately.
In each year's data set, group by artistID, sum the weights, and extract the first ten maximums.

MAPPER

Convert the input text to Key Value pair output. Where Key is NullWritable and value is the whole content. Setting the key to null helps all records enter the same reducer task, so that we can find the global top 10.

```java
public class TopTenMapper extends Mapper<LongWritable, Text, NullWritable, Text>{

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, NullWritable, Text>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        String[] tokens = value.toString().split("\\t");
        String year = tokens[0];
        String artitsId = tokens[1];
        int count = Integer.parseInt(tokens[2]);
//      Text outValue = new Text(year + "\t" + artitsId + "\t" + count);
        context.write(NullWritable.get(), value);
    }
}
```

REDUCER

This reducer has two tasks: summing, and finding the top ten.

1. A Map is used to store the sum of the weights of each artist. The format of the map is: Map <artistID, SUM (weight)>. Sum the weights of each artist after grouping, and save the weights of all artists in a map, so as to facilitate the superposition of the corresponding artist weights. After that, we can know how many times each artist has been listened to.
2. Because before we get the sum of the weight of all artists, we can't know whose sum is the largest. This step must be performed after all artist sums are known. Transfer the entries in the map to the treemap to get the first ten entries. If the size of the TreeMap is greater than 10, the minimum value is evicted. After all the data in the map has been transferred to the treemap, we have the final result.

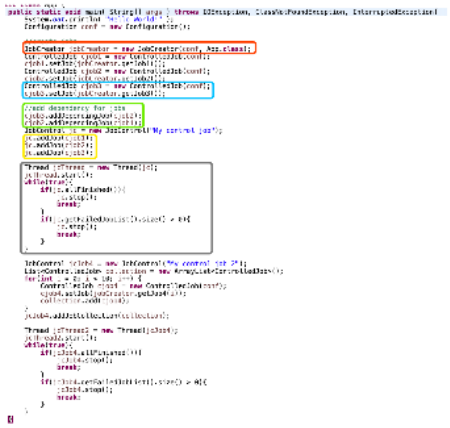
```
protected void reduce(NullWritable key, Iterable<Text> values,
        Reducer<NullWritable, Text, NullWritable, Text>.Context context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    report = new TreeMap<Integer, String>();
    map = new HashMap<Integer, Integer>();
    //get the sum of all the artist
    String year = "";
    for(Text value : values) {
        String[] tokens = value.toString().split("\t");
        year = tokens[0];
        int artistID = Integer.parseInt(tokens[1]);
        int weight = Integer.parseInt(tokens[2]);
        if(!map.containsKey(artistID)) {
            map.put(artistID, 0);
        }
        map.put(artistID, map.get(artistID) + weight);
    }
    //find the top ten
    for(int artistID : map.keySet()) {
        String value = year + "\t" + artistID;
        report.put(map.get(artistID), value);
        if(report.size() > 10) {
            report.remove(report.firstKey());
        }
    }

    for(int count : report.descendingKeySet()) {
        String value = report.get(count) + "\t" + count;
        context.write(NullWritable.get(), new Text(value));
    }
}
```

MAIN CLASS

The 4 jobs described previously depend on each other and are executed in order, so I use JobControl to control all jobs. The 4 jobs wait for the previous job to complete before they can start in the order of 1, 2, 3, and 4. Among them, 10 jobs in job4 are performed at the same time, and the ten most popular songs are searched for data of 10 years.
I used a Jobcreater object to control the generation of all jobs. Here are the jobs created in Jobcreater.

JobContrl is controlled by threads. Greatly improved the efficiency of implementation. The following code is used to enable jobs and set the dependency for each job.



RESULT

I got 10 files representing statistics for 10 years.

drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:01	0	0 B	0	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:01	0	0 B	1	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:02	0	0 B	2	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:02	0	0 B	3	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:03	0	0 B	4	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:03	0	0 B	5	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:03	0	0 B	6	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:04	0	0 B	7	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:04	0	0 B	8	
drwxr-xr-x	lvyan	supergroup	0 B	Dec 11 22:05	0	0 B	9	

Take year 2011 as an example.

year	artistID	sum(weight)
2011	792	324663
2011	289	237948
2011	89	119018
2011	67	99608
2011	344	83470
2011	2548	64652
2011	679	62520
2011	300	60620
2011	1246	51906
2011	333	50517

WAY 2: HIVE

Perform the same logic in Hive. The logic in hive is much simpler.

1. Import data into hive data warehouse. User_artist and user_time are two tables I will use to find the top 10 artists for each year.
    - Tables

```
hive> show tables;
OK
crimedata
user_artist
user_time
Time taken: 0.54 seconds, Fetched: 3 row(s)
hive>
```

- user_artist

```
hive> describe user_artist;
OK
userid                  string
artistid                string
weight                  int
Time taken: 0.186 seconds, Fetched: 3 row(s)
```

- user_time

```
hive> describe user_time;
OK
userid                  string
artistid                string
tagid                   string
day                     string
month                   string
year                    int
Time taken: 0.084 seconds, Fetched: 6 row(s)
hive>
```

2. Perform the following database operations on hive. These operations include:
   - Extracting the required columns from user-time
   - Removing duplicate records
   - Joining two data sets
   - Grouping the data sets according to year and artistid
   - Summing weights on each group
   - Partitioning the data according to year
   - Sort on each partition according to the sum of the weights in each group
   - Select the first ten elements after sorting

```
hive> with temp_year as(
    > select distinct userid,  artistid, year
    > from user_time),
    > temp as(
    > select ut.year, ua.artistid,
    > sum(ua.weight) as totalTimes,
    > rank() over(partition by ut.year order by sum(ua.weight) desc) as rk
    > from  user_artist ua
    > join temp_year ut
    > on ua.userid = ut.userid and ua.artistid = ut.artistid
    > group by ua.artistid, ut.year
    > )
    > select year, artistid, totalTimes
    > from temp
    > where rk < 11
    > order by year, rk;
```

3. Get the following results (part). Obviously, the results of hive and map reduce are the same, but the process is much simpler.

year	artistID	sum(weight)

```
2010    289     492799
2010    701     447493
2010    89      334591
2010    792     327828
2010    292     326014
2010    72      192393
2010    344     183717
2010    67      173119
2010    163     169955
2010    288     148769
2011    792     324663
2011    289     237948
2011    89      119018
2011    67      99608
2011    344     83470
2011    2548    64652
2011    679     62520
2011    300     60620
2011    1246    51906
2011    333     50517
```

WAY 3: PIG

Execute the following statements in pig:

1. Load time data

   user_time = load '/Users/lvyan/Downloads/Data/user_taggedartists.dat' using PigStorage('\t') as (userid, artistid, tagid, day, month, year);

2. Extract userid, artistid and year from time

   datauser_time = foreach user_time generate userid, artistid, year;

3. Deduplication of data

   user_time = distinct user_time;

4. Load count dataset

   user_artist = load '/Users/lvyan/Downloads/Data/user_artists.dat' using PigStorage('\t') as (userid, artistid, weight:int);

5. Join the two data sets according to userid and artistid

   user_weight_year = join user_artist by (userid, artistid), user_time by (userid, artistid);

6. Remove userid column

   user_weight_year = foreach user_weight_year generate year, user_artist::artistid, weight;

7. Process 10 years of data separately (take 2011 as an example)
   a. Extracting data for 2011

      data_2011 = filter user_weight_year by year == 2011;

   b. Group by artistID

group_2011 = group data_2011 by artistid;

c. Get sum

sum_2011 = foreach group_2011 generate $0, SUM($1.weight);

d. Sort the record by sum in descending order

order_2011 = order sum_2011 by $1 desc;

8. Find the top 10

top_2011 = limit order_2011 10;

9. Get the results, take 2011 as an example.

artistID	sum(weight)
792	324663
289	237948
89	119018
67	99608
344	83470
2548	64652
679	62520
300	60620
1246	51906
333	50517

PERFORMANCE COMPARISON

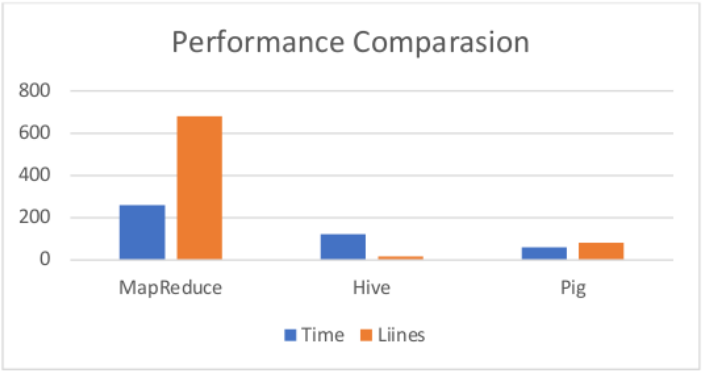
TIME:

The execution time of pig is 60s, the execution time of hive is 123 seconds, and the execution time of Map reduce is 260s.

LINES:

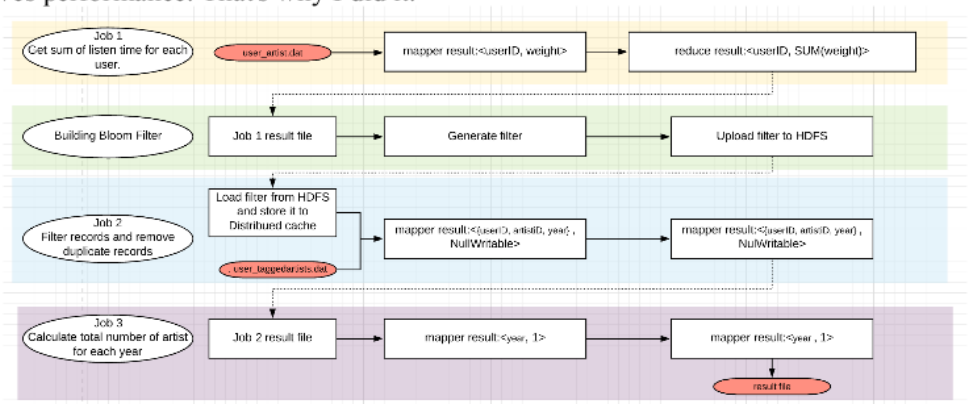
The code of pig is 80 lines, the code of hive is 16 lines, and the code of Map reduce is 680 lines

The conclusion is that both the pig and hive compilers have good performance optimizations for map reduce, especially the optimization of pig is more significant. But because pig operates on data transmission, the logic is more similar to map reduce, and the logic and code are more complicated than hive.



TOTAL NUMBER OF SONGS PLAYED EACH YEAR & BLOOM FILTER & DISTRIBUTED CACHE

Calculated the number of songs played yearly on this platform.
Evaluating the popularity of a platform should be based on how many regular users it has. How to define a regular user? The songs played are limited to users who have accumulated more than 500 listening times, they are regular users.
But the total number of times the user listens needs to calculate the sum of weight in user_artist.dat according to userid. And the number of songs is stored in the user_taggedartists.dat.
In general, we can directly join two data sets. But a better way is to train a bloom filter. At the same time, this is a very large range and very generalized statistics. Therefore, even if the bloom filter has some positive false results, it is still acceptable. Especially when analyzing the change in the popularity of this platform over time, the trend of change will not be affected when the positive false rate is unchanged. So in this case, the positive false rate will not affect the final analysis result, so the bloom filter can be applied here.
First calculate the total number of times each user has listened. Then select only those users who have listened more than 500 times and add them to the filter. In order to apply this filter to every mapper and each reducer, the trained filter is stored in the distributed cache of HDFS. After that, in the setup method in the mapper, load the filter from the distributed cache, and that's it.
This greatly saves memory consumption, simplifies the complex logic when joining, and improves performance. That's why I did it.



BLOOM FILER

Used Hadoop's Bloom Filter. Because the principle of filtering is users whose total number of times of listening to songs is more than 500 times, it is necessary to obtain the sum of the number of times each user listens to songs. So before I built the filter, I used a map reduce to calculate how many times a user listened to songs.

PHASE 1: GET SUM

Use a MapReduce to count the total number of times each user listens to songs.

MAPPER

Get data from user_artist.dat. Extracting userid and the weight from this data set. The emitted key is userid, and the emitted value is weight.

```
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.map(key, value, context);
    if(key.get() == 0) {
        return;
    }
    String[] tokens = value.toString().split("\\t");
    String userID = tokens[0];
    int times = Integer.parseInt(tokens[2]);
    context.write(new Text(userID), new IntWritable(times));
}
```

REDUCER

Calculating the total number of weights for each user. And generate the result.

```
@Override
protected void reduce(Text key, Iterable<IntWritable> values,
        Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    int sum = 0;
    for(IntWritable v : values) {
        sum += v.get();
    }
    context.write(key, new IntWritable(sum));
}
```

RESULT:

Key is userid, value is the sum of weight for this user.

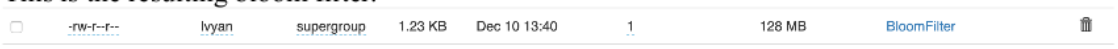
UserID	SUM(weight)
10	28523
100	10358
1001	52320
1002	5740
1003	19647
1004	249
1005	38211
1006	23759
1007	4181
1008	10306
1009	2919
101	58107
1010	273
1011	40712
1012	12826
1013	12
1014	471

PHASE 2: BUILD THE BLOOM FILTER

Read data from precious generated file, and filtering userIDs. If some user didn't listen more than 500 times, he will not be added into the bloom filter. Otherwise, he will be added into the filter.
After building the bloom filter, it is stored into distributed cache in HDFS.
It's the bloom filter, and its parameter settings.

```
public static void main( String[] args ) throws IOException{
      System.out.println( "Hello World!" );
    BloomFilter filter = new BloomFilter(10000, 6, Hash.MURMUR_HASH);
    Path inputFile = new Path(args[0]);
    Path bfFile = new Path(args[1]);
```

This is the resulting bloom filter.



PHASE 3: FILTER DATA BY BLOOM FILTER & REMOVE DUPLICATED RECORDS

MAPPER:

Load file from distributed cache, deserialization the bloom filter. Before generating each result, doing a member test with bloom filter. Only the record that passed the member test, will be emit to the next phase.
Code for loading the filter from distributed cache in mapper:

```
protected void setup(Mapper<LongWritable, Text, UserArtistYearWritable, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.setup(context);
    URI[] files = DistributedCache.getCacheFiles(context.getConfiguration());
    System.out.println("size is:" + files.length);
    System.out.println("Reading Bloom filter from:" + files[0].getPath());
    //open local file for read
    FileSystem fs = FileSystem.get(context.getConfiguration());
    Path path = new Path(files[0].getPath());
    FSDataInputStream is = fs.open(path);
    InputStreamReader inputstreamReader = new InputStreamReader(is);
    filter.readFields(is);
}
```

Code for membership test:

```
@Override
protected void map(LongWritable key, Text value,
        Mapper<LongWritable, Text, UserArtistYearWritable, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.map(key, value, context);
    if(key.get() == 0) {
        return;
    }
    String[] tokens = value.toString().split("\\t");
    String userID = tokens[0];
    if(!filter.membershipTest(new Key(userID.getBytes()))) {
        return;
    }
    String artistID = tokens[1];
    String year = tokens[5];
    UserArtistYearWritable outValue = new UserArtistYearWritable(Integer.parseInt(userID), Integer.parseInt(artistID), Integer.parseInt(year));
    context.write(outValue, NullWritable.get());
}
```

The output format is {userID, artistiD, year} as key, and NullWritable as value. By doing this, can help us to remove the duplicated records.
Here I customized a key writable object to represent a key with three elements. That is UserArtistYearWritable.

```java
public class UserArtistYearWritable implements WritableComparable<UserArtistYearWritable>{

    private int userId;
    private int artistId;
    private int year;

    public UserArtistYearWritable() {
        super();
    }

    public UserArtistYearWritable(int userID, int artisID, int year) {
        super();
        this.userId = userID;
        this.artistId = artisID;
        this.year = year;
    }
```

REDUCER

In reduce phase, only emit one record for each group.
It is the code for reducer which is used to remove the duplicated records.

```java
@Override
protected void reduce(UserArtistYearWritable key, Iterable<NullWritable> valuess,
        Reducer<UserArtistYearWritable, NullWritable, UserArtistYearWritable, NullWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    context.write(key, NullWritable.get());
}
```

MAIN CLASS:

Before creating a job, load the bloom filter that was created in the previous step into the distributed cache.

```java
JobConf conf = new JobConf();
DistributedCache.createSymlink(conf);
DistributedCache.addCacheFile(new URI("hdfs://localhost:8020/FinalProject/BloomFilter#BloomFilter"), conf);
Job job1 = Job.getInstance(conf, "Filter and Remove Duplicated");
```

RESULT:

All members who passed the test, and duplicate records were removed.

UserID	ArtistID	Year
2	52	2009
2	63	2009
2	73	2009
2	94	2009
2	96	2009
2	995	2009
2	3894	2009
2	6160	2009
2	6177	2009
2	9322	2009
3	101	2009
3	101	2010
3	102	2009
3	102	2010
3	106	2010
3	108	2009
3	122	2009
3	128	2010
3	130	2010
3	134	2010
3	137	2009
4	51	2007
4	53	2007
4	64	2007
4	64	2009
4	70	2007
4	72	2007
4	80	2007
4	99	2007
4	151	2007

PHASE 4: GET THE SUM OF SONGS EACH YEAR

MAP:

Only the year information of the input data set is extracted. Since it is a counting problem, all values are 1.

Pearse the input into this format: key is year, value is 1.

```
@Override
protected void map(LongWritable key, Text value,
        Mapper<LongWritable, Text, IntWritable, IntWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    String[] tokens = value.toString().split("\\t");
    int userID = Integer.parseInt(tokens[0]);
    int year = Integer.parseInt(tokens[2]);

    context.write(new IntWritable(year), new IntWritable(1));
}
```

REDUCE:

Get the sum of each year. This is a count of the number of songs played each year.

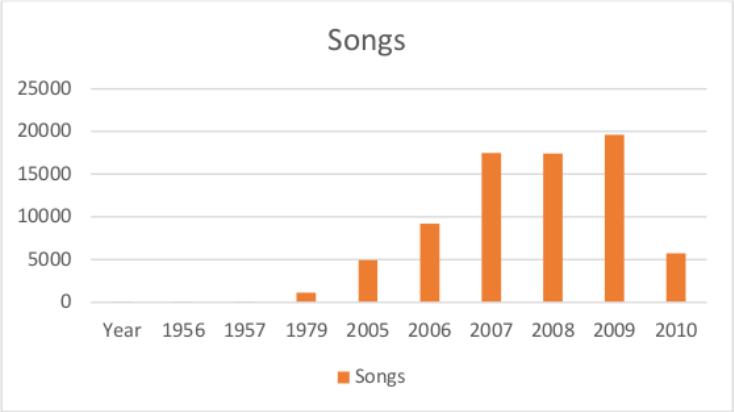
```
@Override
protected void reduce(IntWritable key, Iterable<IntWritable> values,
        Reducer<IntWritable, IntWritable, IntWritable, IntWritable>.Context context)
        throws IOException, InterruptedException {
    int sum = 0;
    for(IntWritable v : values) {
        sum += v.get();
    }
    context.write(key, new IntWritable(sum));
}
```

RESULT:

Year	Count
1956	2
1957	1
1979	1
2005	1156
2006	4911
2007	9185
2008	17510
2009	17434
2010	19649
2011	5771

CONCLUSION:

You can see the change in popularity of this platform. It peaked between 2007 and 2009, and has declined since 2010.



RECOMMENDING PLAY LIST

MAHOUT:USER-BASED & DB MAP

In real life, people prefer to be recommended songs that match their preferences. This algorithm can make targeted recommendations based on the number of times a user has listened to a song. This algorithm mainly uses the recommender in Mahout for song recommendation. The data set used is user_artists.dat. I use the number of times a user has listened to a song as the preference value. The more times I listen, the more I like the song.

MAHOUT

Pearson Correlations was used to calculate the similarity, and the songs that define the two closest users were recommended, and 3 songs were recommended for each user.
Code for creating the recommender and parameter setting.
In this stage, all the users can be recommended. Each user can get a list of recommending paly list.

```
public class App {
    public static void main(String[] args) throws IOException {
        File file = new File("/Users/lvyan/Downloads/Data/UserBasedRecommendation/result.txt");
        Writer out = new FileWriter(file);
        MapDatabase db = new MapDatabase();
        ConcurrentMap<String, String> map = db.getMap("song.db");
        DB database = db.getDatabase();
        try {
            DataModel model = new FileDataModel(new File("/Users/lvyan/Downloads/Data/test.dat"));
            UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
            UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity, model);
            Recommender recommender = new GenericUserBasedRecommender(model, neighborhood, similarity);
            LongPrimitiveIterator a = model.getUserIDs();
            while(a.hasNext()) {
                long userID = a.nextLong();
                List<RecommendedItem> recommendedItems = recommender.recommend(userID, 3);
                String rec = userID + "\t";
                boolean isRec = false;
                for (RecommendedItem recommendation : recommendedItems) {
                    rec += recommendation.getItemID() + ":";
                    rec +=  map.get(recommendation.getItemID() + "") + "\t";
                    isRec = true;
                }
                if(isRec) {
                    out.write(rec.substring(0,rec.length() - 1) + "\n");
                    out.flush();
                }
            }
        } catch (TasteException ex) {
            System.out.print("TasteException: " + ex.getMessage());
        } catch (IOException ex) {
            System.out.print("IOException: " + ex.getMessage());
        }finally{
            out.close();
            database.close();
        }
    }
}
```

But the current song is represented by id. In order to improve readability, I used DB Map to map artistID to name to enhance readability.

DB MAP

It is the code for creating the db map, and how did I store this database in my disk. By doing this, system can return the artist name as long as you know the artistid.

```
public class MapDatabase {
    DB database;

    public DB getDatabase() {
        return this.database;
    }

    public ConcurrentMap<String, String> getMap(String dbFile) throws NumberFormatException, IOException {
        database = DBMaker.fileDB(dbFile).fileMmapEnable().transactionEnable()
                .closeOnJvmShutdown().make();
        ConcurrentMap<String, String> map = database.hashMap("map", Serializer.STRING, Serializer.STRING).createOrOpen();
        File file = new File("/Users/lvyan/Downloads/Data/artists.dat");
        if(file.isFile() && file.exists()) {
            InputStreamReader read = new InputStreamReader(new FileInputStream(file));
            BufferedReader bufferedReader = new BufferedReader(read);
            String line = null;
            while ((line = bufferedReader.readLine()) != null){
                String[] tokens = line.split("\\t");
                String id = tokens[0];
                String name = tokens[1];
                map.put(id, name);
            }
            bufferedReader.close();
            read.close();
        }

        return map;
    }
}
```

RESULT :

In this result set, artist name is written after the artistid and the artist names come from db map.

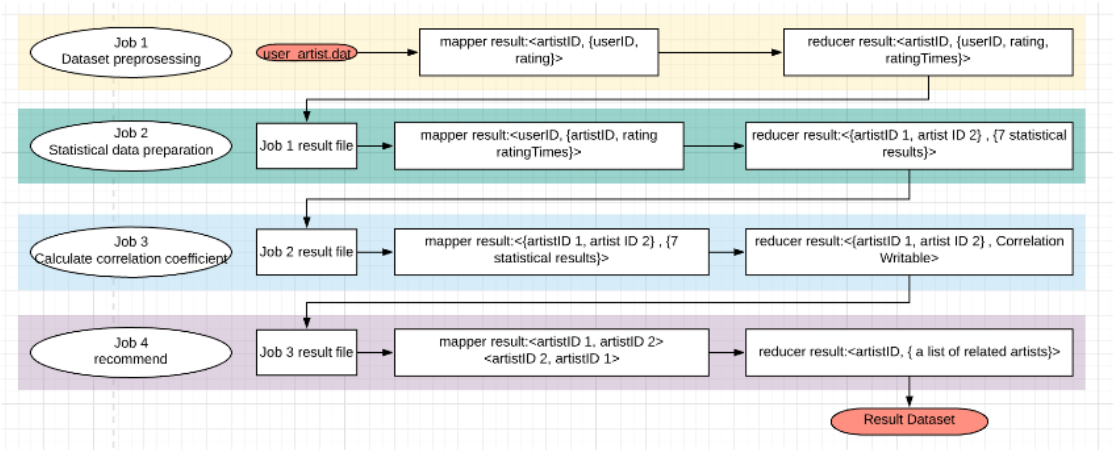
result.txt

```
2       378:Evanescence 966:Nickelback  302:P!nk
4       227:The Beatles 604:Kings of Convenience        618:Belle and Sebastian
5       289:Britney Spears
6       289:Britney Spears      288:Rihanna     89:Lady Gaga
7       726:Apocalyptica
9       475:Eminem
10      707:Metallica   190:Muse        706:AC/DC
11      972:t.A.T.u.
14      159:The Cure    1412:Led Zeppelin       163:Pink Floyd
15      72:Depeche Mode 190:Muse        163:Pink Floyd
16      295:Beyoncé     292:Christina Aguilera  298:Lily Allen
17      475:Eminem      377:Linkin Park
20      333:Avril Lavigne
21      328:David Guetta
24      982:Foo Fighters         962:The Cardigans       183:Jamiroquai
27      288:Rihanna     289:Britney Spears      89:Lady Gaga
28      157:Michael Jackson     67:Madonna
29      56:Daft Punk    65:Coldplay      71:Dido
31      234:Nirvana     230:Green Day
32      89:Lady Gaga    461:Miley Cyrus 466:Ke$ha
33      851:Manic Street Preachers      72:Depeche Mode 599:David Bowie
35      959:Queen       163:Pink Floyd  1412:Led Zeppelin
37      190:Muse
38      163:Pink Floyd
39      503:In Flames   816:A Day to Remember   804:Darkest Hour
40      9:Combichrist   930:Nightwish
41      227:The Beatles 7:Marilyn Manson        504:HIM
42      69:Sade
44      159:The Cure
45      328:David Guetta
```

MAPREDUCE: CONTENT-BASED RECOMMENDATION

Analyze the relevance of the two songs according to the user's preference for the two songs. If artist 1 and artist 2 are highly correlated, then it can be assumed that users who have heard artist 1 will also like artist 2. So, I decided to recommend it by comparing the correlation coefficients of the two songs.

This algorithm uses 4 jobs and is a bit complicated. I use the figure below to roughly show the basic process and the data transmission.



PHASE 1 : DATA PREPROCESSING

For data preprocessing, the dataset to be used is user_artist.dat.

MAP :

The operation at this stage is to extract userID, artistID and rating from the data set. The output key is artisticID, and the value is {userID, rating} pair. I used a customized value writable to represent the value pairs.

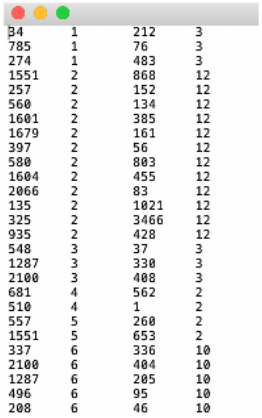
REDUCE :

Calculate the total number of times a song has been rated. The output key is artisitID and value is {userID, rating, ratingTimes} tuples.
The first for loop is used for calculating the total times of rating.
The second loop is used for creating the output value – ArtistRatingSumWritable.

```
@Override
protected void reduce(IntWritable key, Iterable<UserRatingSumWritable> values,
        Reducer<IntWritable, UserRatingSumWritable, IntWritable, ArtistRatingSumWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    int sum = 0;
    List<ArtistRatingSumWritable> list = new ArrayList<ArtistRatingSumWritable>();
    for(UserRatingSumWritable ur : values) {
        sum++;
        sum += ur.getRating();
        int artistID = key.get();
        int rating = ur.getRating();
        ArtistRatingSumWritable outValue = new ArtistRatingSumWritable(artistID, rating, ur.getUserID());
        list.add(outValue);
    }
    IntWritable outKey = new IntWritable(ur.getUserID());
    for(ArtistRatingSumWritable v : list) {
        IntWritable outKey = new IntWritable(v.getSum());
        v.setSum(sum);
        context.write(outKey, v);
    }
}
```

RESULT

artisticID	userID	rating	Rating times



```
34      1       212     3
785     1       76      3
274     1       483     3
1551    2       868     12
257     2       152     12
560     2       134     12
1601    2       385     12
1679    2       161     12
397     2       56      12
580     2       803     12
1604    2       455     12
2066    2       83      12
135     2       1021    12
325     2       3466    12
935     2       428     12
548     3       37      3
1287    3       330     3
2100    3       408     3
681     4       562     2
510     4       1       2
557     5       260     2
1551    5       653     2
337     6       336     10
2100    6       404     10
1287    6       205     10
496     6       95      10
208     6       46      10
```

PHASE 2: CALCULATE STATISTICS

MAP:

Parse data from the input data set and split it into the following form: key is userID and value is {ArtistID, rating, ratingTimes}, value is of ArtistRatingSumWritable type.

REDUCE :

This is the data preparation phase. There are two main steps. Next we will compare the similarity between the two artists based on the calculation result of this step.
Firstly, combine all songs that have been rated by the same user in pairs to prepare data for calculating correlation. This method is used for generating artist pairs.

```
private List<List<ArtistRatingSumWritable>> generateUniqueCombinations(List<ArtistRatingSumWritable> valueList) {
    // TODO Auto-generated method stub
    List<List<ArtistRatingSumWritable>> list = new ArrayList<List<ArtistRatingSumWritable>>();
    for(int i = 0; i < valueList.size() - 1; i++) {
        ArtistRatingSumWritable v1 = valueList.get(i);
        ArtistRatingSumWritable v2 = valueList.get(i + 1);
        if(v1.getArtistID() < v2.getArtistID()) {
            System.out.println("++++++++++++++++"+valueList.size());
            List<ArtistRatingSumWritable> pair = new ArrayList<ArtistRatingSumWritable>();
            pair.add(v1);
            pair.add(v2);
            list.add(pair);
        }
    }
    return list;
}
```

Secondly, calculate the following data between each two songs:
The rating of song 1, the rating of song 2, the number of ratings of song 1, the number of ratings of song 2, the product of ratings of song 1 and song 2, the square of song 1 rating, and the square of song 2 rating. The output at this stage is: key is {artitist 1, artitist 2}, and value is a String made up of 7 numbers just calculated.

```
for(List<ArtistRatingSumWritable> pair : list) {
    ArtistRatingSumWritable a1 = pair.get(0);
    ArtistRatingSumWritable a2 = pair.get(1);
    Text reducerKey = new Text(a1.getArtistID() + "," + a2.getArtistID());
    int ratingProduct = a1.getRating() * a2.getRating();
    int rating1Squared = a1.getRating() * a1.getRating();
    int rating2Squared = a2.getRating() * a2.getRating();
    String out = a1.getRating() + "," + a1.getSum() + "," + a2.getRating() + "," +
            a2.getSum() + "," + ratingProduct + "," + rating1Squared + "," + rating2Squared;
    Text reducerValue = new Text(out);
    context.write(reducerKey, reducerValue);
}
```

RESULT

The results show each pair of artists and their statistical relationship.

part-r-00000 (1)

```
72,74   2619,282,2547,1,6670593,6859161,6487209
73,97   2584,7,1337,80,3454808,6677056,1787569
67,94   3301,429,1373,2,4532273,10896601,1885129
94,95   1373,2,1363,13,1871399,1885129,1857769
65,89   3579,369,1519,611,5436501,12809241,2307361
64,66   3644,71,3312,19,12068928,13278736,10969344
66,69   3312,19,2720,29,9008640,10969344,7398400
63,96   3735,41,1342,42,5012370,13950225,1800964
61,70   3923,25,2686,85,10537178,15389929,7214596
70,92   2686,85,1411,1,3789946,7214596,1990921
60,87   4147,2,1559,1,6465173,17197609,2430481
59,86   4337,107,1594,18,6913178,18809569,2540836
85,90   1638,21,1471,6,2409498,2683044,2163841
51,84   13883,111,1740,41,24156420,192737689,3027600
52,99   11690,23,1330,49,15547700,136656100,1768900
53,58   11351,75,4616,82,52396216,128845201,21307456
54,91   10300,18,1438,11,14811400,106090000,2067844
77,100  2120,43,1315,19,2787800,4494400,1729225
93,98   1407,16,1332,22,1874124,1979649,1774224
55,76   8983,298,2382,15,21397506,80694289,5673924
107,145 269,24,68,1,18292,72361,4624
145,149 68,1,66,1,4488,4624,4356
149,150 66,1,65,1,4290,4356,4225
114,127 167,1,89,1,14863,27889,7921
127,128 89,1,89,1,7921,7921,7921
128,129 89,1,86,1,7654,7921,7396
129,141 86,1,70,3,6020,7396,4900
123,142 104,1,70,1,7280,10816,4900
109,131 215,1,84,1,18060,46225,7056
131,143 84,1,70,1,5880,7056,4900
```

PHASE 3: CALCULATE CORRELATION COEFFICIENT

To calculate the correlation between two artists, Pearson Correlation, Cosine Correlation and Jaccard Correlation are used here to measure the correlation between songs.

MAP

Parse the data, keys are in the {artist1, artist2} format and value in the form of rating-related calculated values. Rating-related calculated values are what we calculating in the precious step.

REDUCE

Correlation is calculated for each artist pair, and the result is recorded in the Correlation Writable object.

Code for calculating the three kinds of correlations.

```
static double calculatePearsonCorrelation(double size, double dotProduct, double rating1Sum, double rating2Sum,
        double rating1NormSq, double rating2NormSq) {
    double numerator = size * dotProduct - rating1Sum * rating2Sum;
    double denominator = Math.sqrt(size * rating1NormSq - rating1Sum * rating1Sum)
            * Math.sqrt(size * rating2NormSq - rating2Sum * rating2Sum);
    return numerator / denominator;
}

static double calculateCosineCorrelation(double dotProduct, double rating1Norm, double rating2Norm) {
    return dotProduct / (rating1Norm * rating2Norm);
}

static double calculateJaccardCorrelation(double inCommon, double totalA, double totalB) {
    double union = totalA + totalB - inCommon;
    return inCommon / union;
}
```

Data parsing and calculation process in reducer, all the correlations are stored into CorrelationWritble objects:

```
protected void reduce(Text key, Iterable<Text> values,
        Reducer<Text, Text, Text, CorrelationWritable>.Context context) throws IOException, InterruptedException {
    // TODO Auto-generated method stub
/   super.reduce(arg0, arg1, arg2);
    int groupSize = 0;
    int dotProduct = 0;
    int rating1Sum = 0;
    int rating2Sum = 0;
    int rating1NormSq = 0; // sum of rating1Squared
    int rating2NormSq = 0;
    int maxNumOfumRaters1 = 0; // max of numOfRaters1
    int maxNumOfumRaters2 = 0;

    for (Text value : values) {
        groupSize++;
        // parse the value
        String[] tokens = value.toString().split(",");
        int rating1 = Integer.parseInt(tokens[0]);
        int numOfRaters1 = Integer.parseInt(tokens[1]);
        int rating2 = Integer.parseInt(tokens[2]);
        int numOfRaters2 = Integer.parseInt(tokens[3]);
        int ratingProduct = Integer.parseInt(tokens[4]);
        int rating1Squared = Integer.parseInt(tokens[5]);
        int rating2Squared = Integer.parseInt(tokens[6]);

        dotProduct += ratingProduct;
        rating1Sum += rating1;
        rating2Sum += rating2;
        rating1NormSq += rating1Squared;
        rating2NormSq += rating2Squared;

        if (numOfRaters1 > maxNumOfumRaters1) {
            maxNumOfumRaters1 = numOfRaters1;
        }

        if (numOfRaters2 > maxNumOfumRaters2) {
            maxNumOfumRaters2 = numOfRaters2;
        }
    }

    double pearson = calculatePearsonCorrelation(groupSize, dotProduct, rating1Sum, rating2Sum, rating1NormSq,
            rating2NormSq);

    double cosine = calculateCosineCorrelation(dotProduct, Math.sqrt(rating1NormSq), Math.sqrt(rating2NormSq));

    double jaccard = calculateJaccardCorrelation(groupSize, maxNumOfumRaters1, maxNumOfumRaters2);

    CorrelationWritable outValue = new CorrelationWritable(pearson, cosine, jaccard);

    context.write(key, outValue);
}
```

Implementation of CorrealtionWritable object:

```
public class CorrelationWritable implements Writable{

    private double pearson;
    private double cosine;
    private double jaccard;

    public CorrelationWritable(){
        super();
    }

    public CorrelationWritable(double i, double j, double k) {
        super();
        this.pearson = i;
        this.cosine = j;
        this.jaccard = k;
    }
```

RESULT

The results show each pair of artists and their correlation coefficients.

```
1,1162  NaN     1.0     0.1
1,1713  NaN     1.0     0.016666666666666666
1,903   NaN     1.0     0.005076142131979695
10,10867        NaN     1.0     0.1111111111111111
10,11   -1.0    0.8270945617241334      0.15384615384615385
10,1297 NaN     1.0     0.05555555555555555
10,1400 NaN     1.0     0.006622516556291391
10,24   NaN     1.0     0.08333333333333333
10,3774 NaN     1.0     0.041666666666666664
10,5180 NaN     1.0     0.07692307692307693
10,758  NaN     1.0     0.05263157894736842
100,1000        NaN     1.0     0.038461538461538464
100,10652       NaN     1.0     0.05
100,1416        NaN     1.0     0.018867924528301886
100,1556        NaN     1.0     0.017241379310344827
100,167 NaN     1.0     0.015384615384615385
100,18225       NaN     1.0     0.05263157894736842
100,1895        NaN     1.0     0.03125
100,193 NaN     1.0     0.014285714285714285
100,2938        NaN     1.0     0.021276595744680851
100,441 NaN     1.0     0.006289308176100629
100,4885        NaN     1.0     0.05
100,712 NaN     1.0     0.016666666666666666
100,8998        NaN     1.0     0.047619047619047616
100,994 NaN     1.0     0.03333333333333333
1000,1001       1.0     0.9987136677136065      0.020202020202020204
1000,1156       NaN     1.0     0.05263157894736842
1000,18302      NaN     1.0     0.125
1000,3183       NaN     1.0     0.06666666666666667
```

PHASE 4: RECOMMENDATION

Recommended songs, I used Correlation Writable objects calculated by the previous step.

MAP:

Filter out songs that don't meet the recommendations. The filtering principle is Pearson Correlation >= 0.5, Cosine Correlation <= 0.5 and Jaccard Correlation >= 0.5. If any one of the conditions is met, the artist can be recommended.

Mapper will produce a pair of outputs. Because if the two songs are highly correlated, artist2 can be recommended when artist1 exists. At the same time, artist1 should be recommended when artist2 exists.

The output is: key is artist1, value is artist 2 & key is artist2, value is artist 1. The mapper produces a pair of outputs for each input.

```java
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, IntWritable, IntWritable>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.map(key, value, context);
    String[] tokens = value.toString().split("\\t");
    String[] artists = tokens[0].split(",");
    double p = Double.parseDouble(tokens[1]);
    double c = Double.parseDouble(tokens[2]);
    double j = Double.parseDouble(tokens[3]);
    int a1 = Integer.parseInt(artists[0]);
    int a2 = Integer.parseInt(artists[1]);
    if((p >= 0.5 || c <= 0.5 || j >= 0.5) && a1 != a2) {
        context.write(new IntWritable(a1), new IntWritable(a2));
        context.write(new IntWritable(a2), new IntWritable(a1));
    }
}
```

REDUCE:

Outputs a list of all the recommended artists of the same artist. These artists has a strong correlation with it.

```java
@Override
protected void reduce(IntWritable key, Iterable<IntWritable> values,
        Reducer<IntWritable, IntWritable, IntWritable, Text>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    String out = "";
    for(IntWritable t : values) {
        out += t.get() + ",";
    }
    out = out.substring(0, out.length() - 1);
    context.write(key, new Text(out));
}
```

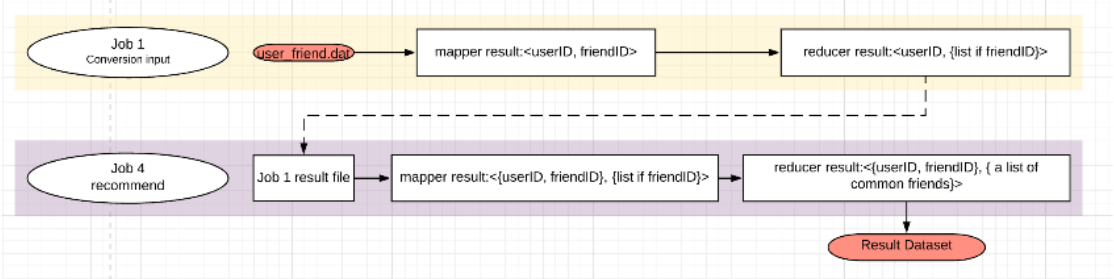
RESULT

Formed a recommendation list for each song. If the user has heard artist 4, then the song corresponding to artist 4 can be recommended to him. The recommended song list according to all songs the user has heard is the recommended playlist.

```
4       7624,9244
7       360,857,1048,1406,1787,233
13      1714
30      1083
34      36
36      2043,34
45      537,498
51      227,159,157,874,875,986,72
53      1090,182
55      1580,1505,972,913,903,797,679,6716,67,65,58,528,518,498,352,329,320,314,302,301,300,299,298,293,289,280,279,257,2523,2094,1934
56      225,1380,59,190,1414
58      238,55,89,59,64,389,295
59      599,1013,1513,1633,2938,3280,905,998,56,58
64      58,1109,70
65      302,300,2977,294,289,89,233,230,228,212,2083,6817,203,906,199,190,1901,1042,1004,182,1048,1249,1479,166,1510,163,1511,498,969,527,55,533,566,67,683,706,493,3666,3293,3200,306
67
97,959,903,89,72,706,701,679,674,646,601,543,542,523,511,498,466,465,459,349,333,329,328,325,318,316,314,311,3081,306,309,298,297,295,294,293,292,291,289,288,283,229,2094,1897,1708,1
61,1452,1428,1246,1104,1045,65,966,55,98,972
68      993,1001
69      5787,1116,1427
70      64
71      182,298
72      1099,8582,706,67,1131,1249,1983,190,187,173,159,152,1379,51,533,89,98,511,440,863,4186,3988,227,1014
75      1151,997
76      1001
81      173,618,439
84      238,546
85      511
86      689
88      424,226,486,190,163,679,207
89
1934,229,257,265,285,291,,190,172,295,1684,298,1623,300,1601,301,157,302,306,308,311,317,320,3200,326,328,331,332,333,344,349,3674,412,461,464,72,466,523,1455,526,1449,528,531,537,538
,5416,546,599,646,679,680,683,686,693,701,903,959,220,1098,1037,1032,65,58,67,292,2014,198,2083
97      903,302,67
98      67,72
```

FIND COMMON FRIENDS

Find the common friends of the two users from the current user's friend list. For some social software users, before following a new friend, they often want to know how many friends they have in common. The more friends in common between two users, the more likely they are to become friends. This algorithm is used to tell the user how many common friends he has with other users.



PHASE 1: CONVERT DATA FORMAT

Conversion input. The format of the original data set is a pair consisting of {user1, friend}, which needs to be converted into the form of {user1, list <friend>} at this stage.

MAP :

Parse the input dataset, and output data format should be: key is userID, value is friendID.

REDUCE :

Join all friends of the same user into a list.

OUTPUT :

```
10      1196,1986,1562,1536,1498
100     1258
1001
120,79,2092,2031,1954,1898,1866,1813,1803,1742,1646,1632,1620,1567,1563,1525,1515,1455,1445,1423,1365,1290,1190,1179,1132,1106,1086,1056,941,930,852,851,847,815,809,795,769,743,691,690,621,571,470,451,437,415,414,413,399,335,332,328,310,256,227,207,179,175,162,145,143,136
1002    910,852,847,778,702,617,7,49,210,232,380,873,385,405,474,498,502,561,2031,1850,1780,1761,1725,1642,1625,1572,1555,1533,1517,1311,1301,1126,1102,1034,1027
1003    1316,224,1818
1004    1632,1504,1194,212,831,941,1522,1956,1789,1719
1005    670,70,1574,1025,895,789
1006    569,723,933,1063,1219,1261,1665,1795,1830,655,61,475
1007    847,701,87,162,163,49
1008    1123,416
1009    1259,2069
101     1161,1267,1388,1519,1743,2027,2070,57,198,202,207,532,964,986,1088
1010    1210,1202,831,330,73,1845,1944,1454
1011    410,691,862,946,1146,1300,1319,1410,1476,1662,1664,1773,1895,247,499,348,388
1012    1316
```

PHASE 2: FIND COMMON FRIENDS

This phase is the phase of finding two friends in common.

MAP :

Traverse the user's friend list to form a {user, friend} pair as the key, and value is all the friends of the user. The difference between this output and the original data is that the user's friend knows who the user's other friends are, and these people are his potential friends. Similarly, friend of friend is also potential friend of user. In order to ensure that the same {user, friend} pair can be assigned to the same group, the userID and friendID in the key must be in the order of userID <friendID.

```java
@Override
protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.map(key, value, context);
    String[] tokens = value.toString().split("\\t");
    int userID = Integer.parseInt(tokens[0]);
    String[] friends = tokens[1].split(",");
    Text outKey;
    for(String friend : friends) {
        int friendID = Integer.parseInt(friend);
        outKey = getOutKey(userID, friendID);
        if(outKey == null) {
            continue;
        }
        context.write(outKey, new Text(tokens[1]));
    }
}

private Text getOutKey(int userID, int friendID) {
    // TODO Auto-generated method stub
    if(userID == friendID) {
        return null;
    }
    String out = "";
    if(userID < friendID) {
        out = userID + "," + friendID;
    }else {
        out = friendID + "," + userID;
    }
    return new Text(out);
}
```

REDUCE :

At this stage, there are at least two values for the same {user, friend} pair. Because one from user and one from friend. The intersection of these values is the common friends of the two users.

Code for finding the intersection of two lists:

```
private Set<Integer> intersection(Set<Integer> friends1, Set<Integer> friends2) {
    // TODO Auto-generated method stub
    if(friends1 == null || friends1.isEmpty()) {
        return null;
    }
    if(friends2 == null || friends2.isEmpty()) {
        return null;
    }
    if(friends1.size() < friends2.size()) {
        return intersect(friends1, friends2);
    }else {
        return intersect(friends2, friends1);
    }
}

private Set<Integer> intersect(Set<Integer> smallSet, Set<Integer> largeSet) {
    // TODO Auto-generated method stub
    Set<Integer> result = new TreeSet<Integer>();
    for(Integer x : smallSet) {
        if(largeSet.contains(x)) {
            result.add(x);
        }
    }
    return result;
}
```

Reduce method to get find the common friends:

```
@Override
protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context)
        throws IOException, InterruptedException {
    // TODO Auto-generated method stub
    super.reduce(arg0, arg1, arg2);
    List<Set<Integer>> listOfValue = new ArrayList<Set<Integer>>();
    for(Text value : values) {
        Set<Integer> set = new HashSet<Integer>();
        String[] friendList = value.toString().split(",");
        for(String friend : friendList) {
            set.add(Integer.parseInt(friend));
        }
        listOfValue.add(set);
    }
    Set<Integer> commonFirneds = listOfValue.get(0);
    for(int i = 1; i < listOfValue.size(); i++) {
        commonFirneds = intersection(commonFirneds, listOfValue.get(i));
    }
    context.write(key, new Text(commonFirneds.toString()));
}
```

OUTPUT :

This is a common friend between users.

```
10,1196 [1562]
10,1498 []
10,1530 [1562]
10,1562 [1196, 1530]
10,1986 []
100,1258        []
1001,1056       [256, 414, 691, 795, 941, 1445]
1001,1086       [143, 162, 571, 621, 1455, 1567, 1846]
1001,1106       [145, 256, 691, 749, 851, 1179, 1515]
1001,1132       [162, 256, 414, 415, 451, 1179, 1813]
1001,1179       [1106, 1132, 1515, 1567, 1632]
1001,1198       [162, 179, 310, 941, 1365, 1898]
1001,1290       []
1001,1365       [120, 256, 413, 690, 852, 930, 941, 1198, 1515, 1742, 1803, 1813, 1898]
1001,1423       [145, 851]
1001,1445       [136, 145, 162, 227, 437, 571, 621, 691, 851, 930, 1056, 1813, 1898]
1001,1455       [621, 691, 749, 847, 851, 852, 930, 1086]
1001,1515       [79, 136, 145, 162, 256, 310, 335, 413, 437, 451, 749, 1106, 1179, 1365, 1567, 1803]
1001,1525       [120, 310, 399, 437, 795, 930, 1846]
1001,1563       [691, 1646]
1001,1567       [145, 162, 437, 691, 749, 1086, 1179, 1515, 1632, 1803]
1001,1628       [145, 414, 621, 690, 1742, 1803]
1001,1632       [162, 256, 335, 413, 691, 1179, 1567]
1001,1646       [207, 328, 335, 852, 1563]
1001,1742       [399, 413, 414, 415, 470, 621, 690, 795, 852, 941, 1365, 1628, 1813]
1001,1803       [162, 621, 1365, 1515, 1567, 1628]
```

OPERATION GUIDE

1. Find the most popular songs of the year:
    a. com.bigdata.TopTenForEachYear.App

2. Total number of songs played each year
    a. Get sum: com.bigdata.mrToGetSum.GetSumApp.
    b. Create Bloom Filter: com.bigdata.bloomFilter.App
    c. Get the result：com.bigdata.yearSum.App
3. Recommendation
    a. com.recommendation.MRPhase1App
    b. com.recommendation.MRPhase2App
    c. com.recommendation.MRPhase3App
    d. com.recommendation.MRPhase4App
4. Common friends
    a. com.commonFirends.CommonApp

APPENDIX

PACKAGE COM.BIGDATA.BLOOMFILTER

APP.JAVA

```
package com.bigdata.bloomFilter;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.zip.GZIPInputStream;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;
import org.apache.hadoop.util.hash.Hash;

/**
 * Hello world!
 *
 */
public class App {
    public static void main( String[] args ) throws IOException{
//        System.out.println( "Hello World!" );
            BloomFilter filter = new BloomFilter(10000, 6, Hash.MURMUR_HASH);
            Path inputFile = new Path(args[0]);
            Path bfFile = new Path(args[1]);
            FileSystem fs = FileSystem.get(new Configuration());
//            FileStatus fileStatus = fs.getFileStatus(inputFile);
//            GZIPInputStream gzip = new GZIPInputStream(fs.open(fileStatus.getPath()));
//            InputStreamReader isr = new InputStreamReader(gzip, "ISO-8859-1");
            FSDataInputStream hdfsInStream = fs.open(new Path(args[0]));
```

```
        InputStreamReader reader = new InputStreamReader(hdfsInStream);
        BufferedReader rdr = new BufferedReader(reader);
//      System.out.println("reading  " + fileStatus.getPath());
        String line = "";
        while((line = rdr.readLine()) != null) {
                String[] tokens = line.split("\\t");
                int sum = Integer.parseInt(tokens[1]);
                if(tokens.length < 2) {
                        System.out.println("<2");
                        continue;
                }
                if(sum > 500) {
                        String userID = tokens[0];
                        Key key = new Key(userID.getBytes());
                        filter.add(key);
                }
        }

        System.out.print("Serializing Boolean filter to HDFS at :" + bfFile);
        FSDataOutputStream strm = fs.create(bfFile);
        filter.write(strm);
        strm.flush();
        strm.close();
        System.exit(0);
  }
}
```

PACKAGE COM.BIGDATA.MAHOUTREC

APP.JAVA

```
package com.bigdata.mahoutRec;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.List;
import java.util.concurrent.ConcurrentMap;
import org.apache.mahout.cf.taste.common.TasteException;
import org.apache.mahout.cf.taste.impl.common.LongPrimitiveIterator;
import org.apache.mahout.cf.taste.impl.model.file.FileDataModel;
import org.apache.mahout.cf.taste.impl.neighborhood.NearestNUserNeighborhood;
import org.apache.mahout.cf.taste.impl.recommender.GenericUserBasedRecommender;
import org.apache.mahout.cf.taste.impl.similarity.PearsonCorrelationSimilarity;
import org.apache.mahout.cf.taste.model.DataModel;
import org.apache.mahout.cf.taste.neighborhood.UserNeighborhood;
import org.apache.mahout.cf.taste.recommender.RecommendedItem;
import org.apache.mahout.cf.taste.recommender.Recommender;
import org.apache.mahout.cf.taste.similarity.UserSimilarity;
import org.mapdb.DB;
```

```
public class App {
	public static void main(String[] args) throws IOException {
		File file = new File("/Users/Ivyan/Downloads/Data/UserBasedRecommendation/result.txt");
		Writer out = new FileWriter(file);
		MapDatabase db = new MapDatabase();
		ConcurrentMap<String, String> map = db.getMap("song.db");
		DB database = db.getDatabase();
		try {
			DataModel model = new FileDataModel(new
File("/Users/Ivyan/Downloads/Data/test.dat"));
			UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
			UserNeighborhood neighborhood = new NearestNUserNeighborhood(2, similarity,
model);
			Recommender recommender = new GenericUserBasedRecommender(model,
neighborhood, similarity);
			LongPrimitiveIterator a = model.getUserIDs();
			while(a.hasNext()) {
				long userID = a.nextLong();
				List<RecommendedItem> recommendedItems =
recommender.recommend(userID, 3);
				String rec = userID + "\t";
				boolean isRec = false;
				for (RecommendedItem recommendation : recommendedItems) {
					rec += recommendation.getItemID() + ":";
					rec +=  map.get(recommendation.getItemID() + "") + "\t";
					isRec = true;
				}
				if(isRec) {
					out.write(rec.substring(0,rec.length() - 1) + "\n");
					out.flush();
				}
			}
		} catch (TasteException ex) {
			System.out.print("TasteException: " + ex.getMessage());
		} catch (IOException ex) {
			System.out.print("IOException: " + ex.getMessage());
		}finally{
			out.close();
			database.close();
		}
	}
}
```

MAPDATABSE.JAVA

```
kage com.bigdata.mahoutRec;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.concurrent.ConcurrentMap;
import org.mapdb.DB;
```

```
import org.mapdb.DBMaker;
import org.mapdb.Serializer;

public class MapDatabase {
	DB database;

	public DB getDatabase() {
		return this.database;
	}

	public ConcurrentMap<String, String> getMap(String dbFile) throws NumberFormatException,
IOException {
		database = DBMaker.fileDB(dbFile).fileMmapEnable().transactionEnable()
	.closeOnJvmShutdown().make();
		ConcurrentMap<String, String> map = database.hashMap("map", Serializer.STRING,
Serializer.STRING).createOrOpen();
		File file = new File("/Users/lvyan/Downloads/Data/artists.dat");
		if(file.isFile() && file.exists()) {
			InputStreamReader read = new InputStreamReader(new FileInputStream(file));
			BufferedReader bufferedReader = new BufferedReader(read);
	String line = null;
	while ((line = bufferedReader.readLine()) != null){
	   String[] tokens = line.split("\\t");
	   String id = tokens[0];
	   String name = tokens[1];
	   map.put(id, name);
	}
	bufferedReader.close();
	read.close();
		}

		return map;
	}
}
```

PACKAGE COM.BIGDATA.MRTOGETSUM

GETSUMAPP.JAVA

```
package com.bigdata.mrToGetSum;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
public class GetSumApp {
        public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//      System.out.println( "Hello World!" );
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Triple MapReduce");

        //set driver class
        job.setJarByClass(GetSumApp.class);

        //set mapper output
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        //set input output
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //output key and value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        //set map reduce
        job.setMapperClass(UserTimesMapper.class);
        job.setReducerClass(UserTimesReducer.class);

        Path input = new Path("/FinalProject/Data/user_artists.dat");
        Path output = new Path("/FinalProject/Data/TestSumYearly/timesSum");
//      Path input = new Path("/Users/lvyan/Downloads/Data/user_artists.dat");
//      Path output = new Path("/Users/lvyan/Downloads/Data/timesSum");
//
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);

        FileSystem hdfs = FileSystem.get(conf);
                if(hdfs.exists(output)){
                        hdfs.delete(output, true);
                }

        job.waitForCompletion(true);
  }
}
```

USERTIMESMAPPER.JAVA

```
package com.bigdata.mrToGetSum;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class UserTimesMapper extends Mapper<LongWritable, Text, Text, IntWritable>{
        @Override
```

```
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
IntWritable>.Context context)
                        throws IOException, InterruptedException {
                // TODO Auto-generated method stub
//              super.map(key, value, context);
                if(key.get() == 0) {
                        return;
                }
                String[] tokens = value.toString().split("\\t");
                String userID = tokens[0];
                int times = Integer.parseInt(tokens[2]);
                context.write(new Text(userID), new IntWritable(times));
        }
}
```

USERTIMESREDUCER.JAVA

```
package com.bigdata.mrToGetSum;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class UserTimesReducer extends Reducer<Text, IntWritable,Text, IntWritable>{

        @Override
        protected void reduce(Text key, Iterable<IntWritable> values,
                        Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException,
InterruptedException {
                // TODO Auto-generated method stub
//              super.reduce(arg0, arg1, arg2);
                int sum = 0;
                for(IntWritable v : values) {
                        sum += v.get();
                }
                context.write(key, new IntWritable(sum));
        }
}
```

PACKAGE COM.BIGDATA.TOPTENFOREACHYEAR

APP.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
import org.apache.hadoop.mapreduce.lib.jobcontrol.JobControl;
public class App {
    public static void main( String[] args ) throws IOException, ClassNotFoundException, InterruptedException{
        System.out.println( "Hello World!" );
```

```
Configuration conf = new Configuration();

//create jobs
JobCreator jobCreator = new JobCreator(conf, App.class);
ControlledJob cjob1 = new ControlledJob(conf);
cjob1.setJob(jobCreator.getJob1());
                ControlledJob cjob2 = new ControlledJob(conf);
                cjob2.setJob(jobCreator.getJob2());
                ControlledJob cjob3 = new ControlledJob(conf);
                cjob3.setJob(jobCreator.getJob3());

                //add dependency for jobs
                cjob3.addDependingJob(cjob2);
                cjob2.addDependingJob(cjob1);
                JobControl jc = new JobControl("My control job");
                jc.addJob(cjob1);
                jc.addJob(cjob2);
                jc.addJob(cjob3);

                Thread jcThread = new Thread(jc);
jcThread.start();
while(true){
   if(jc.allFinished()){
      jc.stop();
      break;
   }
   if(jc.getFailedJobList().size() > 0){
      jc.stop();
      break;
   }
}

JobControl jcJob4 = new JobControl("My control job 2");
                List<ControlledJob> collection = new ArrayList<ControlledJob>();
                for(int i = 0; i < 10; i++) {
                        ControlledJob cjob4 = new ControlledJob(conf);
                        cjob4.setJob(jobCreator.getJob4(i));
                        collection.add(cjob4);
                }
                jcJob4.addJobCollection(collection);

                Thread jcThread2 = new Thread(jcJob4);
                jcThread2.start();
while(true){
   if(jcJob4.allFinished()){
      jcJob4.stop();
      break;
   }
   if(jcJob4.getFailedJobList().size() > 0){
      jcJob4.stop();
      break;
   }
}
```

```
    }
}
```

GETWEIGHTMAPPER.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
//user_artists.dat
public class GetWeightMapper extends Mapper<LongWritable, Text, UserArtistWritable, Text>{

        UserArtistWritable outKey;
        Text outValue;
        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, UserArtistWritable,
Text>.Context context)
                        throws IOException, InterruptedException {
                if(key.get() == 0) {
                        return;
                }
                String[] tokens = value.toString().split("\\t");
                int userID = Integer.parseInt(tokens[0]);
                int artistID = Integer.parseInt(tokens[1]);
                outKey = new UserArtistWritable(userID, artistID);
                String count = tokens[2];
                outValue = new Text("B" + count);
                context.write(outKey, outValue);
        }
}
```

GETYEARMAPPER.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
//user_taggedartists.dat
public class GetYearMapper extends Mapper<LongWritable, Text, UserArtistWritable, Text>{
        UserArtistWritable outKey;
        Text outValue;
        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, UserArtistWritable,
Text>.Context context)
                        throws IOException, InterruptedException {
                String[] tokens = value.toString().split("\\t");
                int userID = Integer.parseInt(tokens[0]);
                int artistID = Integer.parseInt(tokens[1]);
                outKey = new UserArtistWritable(userID, artistID);
```

```
                String year = tokens[2];
                outValue = new Text("A" + year);
                context.write(outKey, outValue);
        }
}
```

GROUPCOMPARATOR.JAVA

```
package com.bigdata.TopTenForEachYear;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
public class GroupComparator extends WritableComparator{

        public GroupComparator() {
                super(UserArtistWritable.class, true);
        }

        @Override
        public int compare(WritableComparable a, WritableComparable b) {
                UserArtistWritable ckw1 = (UserArtistWritable) a;
                UserArtistWritable ckw2 = (UserArtistWritable) b;
                int result = ckw1.getUserID()-ckw2.getUserID();
                if(result == 0) {
                        result = ckw1.getArtistID()-ckw2.getArtistID();
                }
                return result;
        }
}
```

JOBCREATOR.JAVA

```
package com.bigdata.TopTenForEachYear;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class JobCreator {

        Configuration conf;
        Class jar;
```

```
FileSystem hdfs;
public JobCreator(Configuration conf, Class jar) throws IOException {
        this.conf = conf;
        this.jar = jar;
        hdfs = FileSystem.get(conf);
}

6ore process the year dataset : remove duplicated records
public Job getJob1() throws IOException {
        Job job = Job.getInstance(this.conf, "Remove dup from year data set");
        7b.setJarByClass(this.jar);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(NullWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(NullWritable.class);
        job.setMapperClass(PreYearMapper.class);
        job.setReducerClass(RemoveDuplicatedReducer.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        Path input = new Path("/FinalProject/Data/user_taggedartists.dat");
        1th output = new Path("/FinalProject/Data/TestChainingJob/year_noDup");
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);
        if(hdfs.exists(output)){
                hdfs.delete(output, true);
        }
        return job;
}

//Get userID, artistID, weight -> join -> year, artistID, weight
public Jo4getJob2() throws IOException {
        Configuration conf = new Configuration();
Job job = Job.getInstance(this.conf, "Join");
job.setJarByClass(this.jar);

job.setMapOutputKeyClass(UserArtistWritable.class);
job.setMapOutputValueClass(Text.class);
//secondary sort
job.setPartitionerClass(PartitionerComparator.class);
job.setGroupingComparatorClass(GroupComparator.class);
job.setSortComparatorClass(SecondarySortComparator.class);
Path countPath = new Path("/FinalProject/Data/user_artists.dat");
1th yearPath = new Path("/FinalProject/Data/TestChainingJob/year_noDup/part-*");
MultipleInputs.addInputPath(job, countPath, TextInputFormat.class, GetWeightMap20.class);
MultipleInputs.addInputPath(job, yearPath, TextInputFormat.class, GetYearMapper.class);
job.setReducerClass(JoinReducer.class);
job.setOutputFormatClass(TextOutputFormat.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(NullWritable.class);
Path output = new Path("/FinalProject/Data/TestChainingJob/afterJoin");
1eOutputFormat.setOutputPath(job, output);
if(hdfs.exists(output)){
                hdfs.delete(output, true);
```

```
		}
		return job;
	}

	//plit by year
	public JobGetJob3() throws IOException {
		Job job = new Job(this.conf, "Partitioning By Year");
		job.setJarByClass(this.jar);
		job.setMapperClass(YearPartitionerMapper.class);
		job.setReducerClass(YearPartitionerReducer.class);
		job.setInputFormatClass(TextInputFormat.class);
		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(Text.class);
		job.setNumReduceTasks(10);
		job.setPartitionerClass(YearPartitioner.class);
		YearPartitioner.setMinLastAccessDateYear(job, 1956);
		Path input = new Path("/FinalProject/Data/TestChainingJob/afterJoin/part-*");
		Path output = new Path("/FinalProject/Data/TestChainingJob/afterPartition");
		FileInputFormat.addInputPath(job, input);
		FileOutputFormat.setOutputPath(job, output);
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(NullWritable.class);
		if(hdfs.exists(output)){
			hdfs.delete(output, true);
		}
		return job;
	}

	//Top 10
	public JobGetJob4(int n) throws IOException {
		Job job = new Job(this.conf, "Get Top 10 For Each Year");
		job.setJarByClass(this.jar);
		job.setMapperClass(TopTenMapper.class);
		job.setReducerClass(TopTenReducer.class);
		job.setInputFormatClass(TextInputFormat.class);
		job.setMapOutputKeyClass(NullWritable.class);
		job.setMapOutputValueClass(Text.class);
		Path input = new Path("/FinalProject/Data/TestChainingJob/afterPartition/part-r-0000" + n);
		Path output = new Path("/FinalProject/Data/TestChainingJob/TopTen/" + n);
		FileInputFormat.addInputPath(job, input);
		FileOutputFormat.setOutputPath(job, output);
		job.setOutputKeyClass(NullWritable.class);
		job.setOutputValueClass(Text.class);
		if(hdfs.exists(output)){
			hdfs.delete(output, true);
		}
		return job;
	}
}
```

JOINREDUCER.JAVA

```
package com.bigdata.TopTenForEachYear;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class JoinReducer extends Reducer<UserArtistWritable, Text, Text, NullWritable> {
	// result if : year, artisid, count

	int min, max;

	@Override
	protected void setup(Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
			throws IOException, InterruptedException {
		min = Integer.MAX_VALUE;
		max = Integer.MIN_VALUE;
	}

	@Override
	protected void reduce(UserArtistWritable key, Iterable<Text> values,
			Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
		// year, count
		List<Integer> yearList = new ArrayList<Integer>();
		List<Integer> countList = new ArrayList<Integer>();
		int year = 0;
		int count = 0;
		for (Text t : values) {
			String value = t.toString();
			if (value.charAt(0) == 'A') {
				year = Integer.parseInt(value.substring(1));
				yearList.add(year);
				min = Math.min(min, year);
				max = Math.max(max, year);
			} else if (value.charAt(0) == 'B') {
				count = Integer.parseInt(value.substring(1));
				countList.add(count);
			}
		}

		for (int y : yearList) {
			String tuple = y + "\t" + key.getArtistID() + "\t";
			for (int c : countList) {
				tuple += c + "";
				context.write(new Text(tuple), NullWritable.get());
			}
		}
```

```
	}
	@Override
	protected void cleanup(Reducer<UserArtistWritable, Text, Text, NullWritable>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
		context.getConfiguration().setInt("min.last.access.date.year", min);
		context.getConfiguration().setInt("num", max - min + 1);
		System.out.println("In the join reduceer: " +
context.getConfiguration().getInt("min.last.access.date.year", 111));
		System.out.println("In the join num: " + context.getConfiguration().getInt("num", 111));
	}
}
```

PARTITIONERCOMPARATOR.JAVA

```
package com.bigdata.TopTenForEachYear;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Partitioner;

public class PartitionerComparator extends Partitioner<UserArtistWritable, Text>{

	@Override
	public int getPartition(UserArtistWritable key, Text value, int num) {
		// TODO Auto-generated method stub
//		return 0;
		return key.getArtistID() % num;
	}

}
```

PREYEARMAPPER.JAVA

```
package com.bigdata.TopTenForEachYear;

import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class PreYearMapper extends Mapper<LongWritable, Text, Text, NullWritable>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text,
NullWritable>.Context context)
			throws IOException, InterruptedException {
		if(key.get() == 0) {
			return;
		}
		String[] tokens = value.toString().split("\\t");
		String userID = tokens[0];
```

```
                String artistID = tokens[1];
                String year = tokens[5];
                String out = userID + "\t" + artistID + "\t" + year;
                context.write(new Text(out), NullWritable.get());
        }
}
```

REMOVEDUPLICATEDREDUCER.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


/remove the duplicated items in the year file I got
public class RemoveDuplicatedReducer extends Reducer<Text, NullWritable,Text, NullWritable>{

        @Override
        protected void reduce(Text key, Iterable<NullWritable> values,
                        Reducer<Text, NullWritable, Text, NullWritable>.Context context) throws IOException,
InterruptedException {
                // TODO Auto-generated method stub
//              per.reduce(arg0, arg1, arg2);
                context.write(key, NullWritable.get());
        }
}
```

SECONDARYSORTCOMPARATOR.JAVA

```
packa m.bigdata.TopTenForEachYear;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
public class SecondarySortComparator extends WritableComparator{
        public SecondarySortComparator() {
                super(UserArtistWritable.class, true);
        }

        @Override
        public int compare(WritableComparable a, WritableComparable b) {
                UserArtistWritable ckw1 = (UserArtistWritable) a;
                UserArtistWritable ckw2 = (UserArtistWritable) b;
                int result = ckw1.getUserID() - ckw2.getUserID();
                if(result == 0) {
                        result = ckw1.getArtistID() - ckw2.getArtistID();
                }
                return result;
        }
}
```

TOPTENMAPPER.JAVA

```
ckage com.bigdata.TopTenForEachYear;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class TopTenMapper extends Mapper<LongWritable, Text, NullWritable, Text>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, NullWritable,
Text>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
		String[] tokens = value.toString().split("\\t");
		String year = tokens[0];
		String artitsId = tokens[1];
		int count = Integer.parseInt(tokens[2]);
//		Text outValue = new Text(year + "\t" + artitsId + "\t" + count);
		context.write(NullWritable.get(), value);
	}
}
```

TOPTENREDUCER.JAVA

```
kage com.bigdata.TopTenForEachYear;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
mport java.util.TreeMap;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class TopTenReducer extends Reducer<NullWritable, Text, NullWritable, Text>{
	//count, "artisID, year"
	TreeMap<Integer, String> report;
	//artistID, count
	M<Integer, Integer> map;
	@Override
	protected void reduce(NullWritable key, Iterable<Text> values,
			Reducer<NullWritable, Text, NullWritable, Text>.Context context) throws IOException,
InterruptedException {
		// TODO Auto-generated method stub
		report = new TreeMap<Integer, String>();
		map = new HashMap<Integer, Integer>();
		//get the sum of all the artist
		String year = "";
		for(Text value : values) {
			String[] tokens = value.toString().split("\t");
			year = tokens[0];
			int artistID = Integer.parseInt(tokens[1]);
			int weight = Integer.parseInt(tokens[2]);
			if(!map.containsKey(artistID)) {
```

```
                        map.put(artistID, 0);
                    }
                    map.put(artistID, map.get(artistID) + weight);
                }
                //find the top ten
                for(int artistID : map.keySet()) {
                    String value = year + "\t" + artistID;
                    report.put(map.get(artistID), value);
                    if(report.size() > 10) {
                        report.remove(report.firstKey());
                    }
                }

                for(int count : report.descendingKeySet()) {
                    String value = report.get(count) + "\t" + count;
                    context.write(NullWritable.get(), new Text(value));
                }
            }
}
```

USERARTISTWRITABLE.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.WritableComparable;
public class UserArtistWritable implements WritableComparable<UserArtistWritable>{

    private int userID;
    private int artistID;

    public UserArtistWritable(){
        super();
    }

    public UserArtistWritable(int userID, int artistID) {
        super();
        this.userID = userID;
        this.artistID = artistID;
    }

    public int getUserID() {
        return userID;
    }

    public void setUserID(int userID) {
        this.userID = userID;
    }

    public int getArtistID() {
        return artistID;
    }
```

```
	public void setArtistID(int artistID) {
		this.artistID = artistID;
	}
	5
	public int compareTo(UserArtistWritable that) {
		// TODO Auto-generated method stub
		double result = this.getArtistID()- that.getArtistID();
		if(result == 0.0) {
			result = this.getUserID() - that.getUserID();
		}
		return (result < 0.0 ? -1 : (result == 0.0 ? 0 : 1));
	}

	5
	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		userID = input.readInt();
		artistID = input.readInt();
	}
	5
	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeInt(userID);
		output.writeInt(artistID);
	}
	1
	public String toString() {
		return this.getArtistID() + "," + this.getUserID();
	}
}
```

YEARPARTITIONER.JAVA

```
2ckage com.bigdata.TopTenForEachYear;
import org.apache.hadoop.conf.Configurable;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.6apreduce.Partitioner;
public class YearPartitioner extends Partitioner<IntWritable, Text> implements Configurable{

	6ivate static final String MIN_LAST_ACCESS_DATE_YEAR = "min.last.access.date.year";
	private Configuration conf = null;
	private int minLastAccessDateYear = 0;

	public Conf43ration getConf() {
		// TODO Auto-generated method stub
		return conf;
	}

	public void setConf(Configuration conf) {
```

```
                // TODO Auto-generated method stub
                this.conf = conf;
                minLastAccessDateYear = conf.getInt(MIN_LAST_ACCESS_DATE_YEAR, 0);
        }

        @Override
        public int getPartition(IntWritable key, Text value, int num) {
                // TODO Auto-generated method stub
//              return 0;
//              return key.get() - minLastAccessDateYear;
                if(key.get() == 1956) {
                        return 0;
                }else if(key.get() == 1957) {
                        return 1;
                }else if(key.get() == 1979) {
                        return 2;
                }else {
                        return key.get() - 2005 + 3;
//                      return 0;
                }
        }
        public static void setMinLastAccessDateYear(Job job, int minLastAccessDateYear) {
                job.getConfiguration().setInt(MIN_LAST_ACCESS_DATE_YEAR, minLastAccessDateYear);
        }
}
```

YEARPARTITIONERMAPPER.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
import java.text.ParseException;
import java.util.Calendar;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

//year, (articleId, weight)
public class YearPartitionerMapper extends Mapper<LongWritable, Text, IntWritable, Text>{

        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, IntWritable,
Text>.Context context) throws IOException, InterruptedException {
                String[] tokens = value.toString().split("\\t");
                int year = Integer.parseInt(tokens[0]);
                String outKey = tokens[1] + "\t" + tokens[2];
                context.write(new IntWritable(year), new Text(outKey));
        }
}
```

YEARPARTITIONERREDUCER.JAVA

```
package com.bigdata.TopTenForEachYear;
import java.io.IOException;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class YearPartitionerReducer extends Reducer<IntWritable, Text, IntWritable, Text>{
	@Override
	protected void reduce(IntWritable key, Iterable<Text> values,
			Reducer<IntWritable, Text, IntWritable, Text>.Context context) throws IOException,
InterruptedExcep54n {
		for(Text t : values) {
			context.write(key, t);
		}
	}
}
```

PACKAGE COM.BIGDATA.YEARSUM

APP.JAVA

```
package com.bigdata.yearSum;
14
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;


/**
 * Hello world!
 *
 */
public class App {
    public static void main( String[] args ) throws IOException, ClassNotFoundException, InterruptedException,
URISyntaxException{
//        System.out.println( "Hello World!" );
        JobConf conf = new JobConf();
        DistributedCache.createSymlink(conf);
        DistributedCache.addCacheFile(new URI("hdfs://localhost:8020/FinalProject/BloomFilter#BloomFilter"),
conf);
        Job job1 = Job.getInstance(conf, "Filter and Remove Duplicated");
```

```
        job1.setJarByClass(App.class);
        job1.setInputFormatClass(TextInputFormat.class);
        job1.setOutputFormatClass(TextOutputFormat.class);
        job1.setMapOutputKeyClass(UserArtistYearWritable.class);
        job1.setMapOutputValueClass(NullWritable.class);
        job1.setOutputKeyClass(IntWritable.class);
    job1.setOutputValueClass(IntWritable.class);

    job1.setMapperClass(FilteringMapper.class);
    job1.setReducerClass(RemoveDuplicatedReducer.class);

    Path input = new Path("/FinalProject/Data/user_taggedartists.dat");
//      Path output = new Path("/HW4/Part6/CountResult");
//      Path input = new Path(args[0]);
    Path output = new Path("/FinalProject/Data/TestSumYearly/AfterRemoveDuplicated");
    FileInputFormat.addInputPath(job1, input);
    FileOutputFormat.setOutputPath(job1, output);

    FileSystem hdfs = FileSystem.get(conf);
                if(hdfs.exists(output)){
                        hdfs.delete(output, true);
                }

    job1.waitForCompletion(true);


    Job job2 = Job.getInstance(conf, "Sum of each year");
    job2.setJarByClass(App.class);
    job2.setInputFormatClass(TextInputFormat.class);
        job2.setOutputFormatClass(TextOutputFormat.class);
        job2.setOutputKeyClass(IntWritable.class);
        job2.setOutputValueClass(IntWritable.class);
        job2.setMapperClass(YearSumMapper.class);
        job2.setReducerClass(YearSumReducer.class);

        Path input2 = new Path("/FinalProject/Data/TestSumYearly/AfterRemoveDuplicated/part-*");
        Path output2 = new Path("/FinalProject/Data/TestSumYearly/YearSum");
//      Path output2 = new Path(args[1]);
    FileInputFormat.setInputPaths(job2, input2);
    FileOutputFormat.setOutputPath(job2, output2);
//      FileSystem hdfs = FileSystem.get(conf);
                if(hdfs.exists(output2)){
                        hdfs.delete(output2, true);
                }
    job2.waitForCompletion(true);

  }
}
```

AVEWRITABLE.JAVA

```
package com.bigdata.yearSum;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class AveWritable implements Writable{

	private int ave;
	private int count;

	public AveWritable() {
		super();
	}

	public AveWritable(int ave,int count) {
		super();
		this.ave = ave;
		this.count = count;
	}

	public int getAve() {
		return ave;
	}

	public void setAve(int ave) {
		this.ave = ave;
	}


	public int getCount() {
		return count;
	}

	public void setCount(int count) {
		this.count = count;
	}

	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		this.ave = input.readInt();
		this.count = input.readInt();
	}

	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeInt(ave);
		output.writeInt(count);
	}

	public String toString() {
```

```
                return this.getAve() + "\t"  + this.getCount();
        }

}
```

FILTERINGMAPPER.JAVA

```
package com.bigdata.yearSum;

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;
//import java.nio.file.FileSystem;

import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;

//计算平均每年每个用户听歌的数量（不是次数）
//user_taggedartists.dat
//需要先去除重复
public class FilteringMapper extends Mapper<LongWritable, Text, UserArtistYearWritable, NullWritable>{

        private BloomFilter filter = new BloomFilter();
        @Override
        protected void setup(Mapper<LongWritable, Text, UserArtistYearWritable, NullWritable>.Context
context)
                        throws IOException, InterruptedException {
                // TODO Auto-generated method stub
//                super.setup(context);
                URI[] files = DistributedCache.getCacheFiles(context.getConfiguration());
                System.out.println("size is:" + files.length);
                System.out.println("Reading Bloom filter from:" + files[0].getPath());
                //open local file for read
                FileSystem fs = FileSystem.get(context.getConfiguration());
                Path path = new Path(files[0].getPath());
                FSDataInputStream is = fs.open(path);
                InputStreamReader inputstreamReader = new InputStreamReader(is);
                filter.readFields(is);
        }

        @Override
        protected void map(LongWritable key, Text value,
```

```
                Mapper<LongWritable, Text, UserArtistYearWritable, NullWritable>.Context context)
                throws IOException, InterruptedException {
        // TODO Auto-generated method stub
//      super.map(key, value, context);
        if(key.get() == 0) {
                return;
        }
        String[] tokens = value.toString().split("\\t");
        String userID = tokens[0];
        if(!filter.membershipTest(new Key(userID.getBytes()))) {
                return;
        }
        String artistID = tokens[1];
        String year = tokens[5];
        UserArtistYearWritable outValue = new UserArtistYearWritable(Integer.parseInt(userID),
Integer.parseInt(artistID), Integer.parseInt(year));
        context.write(outValue, NullWritable.get());
    }

}
```

REMOVEDUPLICATEDREDUCER.JAVA

```
package com.bigdata.yearSum;

import java.io.IOException;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class RemoveDuplicatedReducer extends Reducer<UserArtistYearWritable,
NullWritable,UserArtistYearWritable, NullWritable>{

    @Override
    protected void reduce(UserArtistYearWritable key, Iterable<NullWritable> valuess,
            Reducer<UserArtistYearWritable, NullWritable, UserArtistYearWritable,
NullWritable>.Context context)
            throws IOException, InterruptedException {
        // TODO Auto-generated method stub
//      super.reduce(arg0, arg1, arg2);
        context.write(key, NullWritable.get());
    }
}
```

USERARTISTYEARWRITABLE.JAVA

```
package com.bigdata.yearSum;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;
```

```
public class UserArtistYearWritable implements WritableComparable<UserArtistYearWritable>{

	private int userId;
	private int artistId;
	private int year;

	public UserArtistYearWritable() {
		super();
	}

	public UserArtistYearWritable(int userID, int artisID, int year) {
		super();
		this.userId = userID;
		this.artistId = artisID;
		this.year = year;
	}
	1
	public int getUserId() {
		return userId;
	}

	public void setUserId(int userId) {
		this.userId = userId;
	}

	public int getArtistId() {
		return artistId;
	}

	public void setArtistId(int artistId) {
		this.artistId = artistId;
	}

	public int getYear() {
		return year;
	}

	public void setYear(int year) {
		this.year = year;
	}
	5
	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		this.userId = input.readInt();
		this.artistId = input.readInt();
		this.year = input.readInt();
	}
	5
	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeInt(userId);
		output.writeInt(artistId);
```

```
		output.writeInt(year);
	}
	5
	public int compareTo(UserArtistYearWritable that) {
		// TODO Auto-generated method stub
//		return 0;
		int result = this.getUserId() - that.getUserId();
		if(result == 0) {
			result = this.getArtistId() - that.getArtistId();
		}
		if(result == 0) {
			result = this.getYear() - that.getYear();
		}
		return result;
	}

	public String toString() {
		return getUserId() + "\t" + this.getArtistId() + "\t" + this.getYear();
	}

}
```

USERYEARKEYWRITABLE.JAVA

```
packag 10 m.bigdata.yearSum;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;
public class UserYearKeyWritable implements WritableComparable<UserYearKeyWritable>{

	private int userId;
	private int year;

	public UserYearKeyWritable() {
		super();
	}

	public UserYearKeyWritable(int userID, int year) {
		super();
		this.userId = userID;
		this.year = year;
	}
		16
	public int getUserId() {
		return userId;
	}

	public void setUserId(int userId) {
		this.userId = userId;
	}
```

```
	public int getYear() {
		return year;
	}

	public void setYear(int year) {
		this.year = year;
	}
	5
	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		this.userId = input.readInt();
		this.year = input.readInt();
	}

	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeInt(userId);
		output.writeInt(year);
	}

	5先 userID，再 year
	public int compareTo(UserYearKeyWritable that) {
		// TODO Auto-generated method stub
//		return 0;
		int result = this.getUserId() - that.getUserId();
		if(result == 0) {
			result = this.getYear() - that.getYear();
		}
		return result;
	}
	1
	public String toString() {
		return this.getUserId() + "\t" + this.getYear();
	}

}
```

YEARSUMMAPPER.JAVA

```
package com.bigdata.yearSum;
	2
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

//public 2ss YearSumMapper extends Mapper<LongWritable, Text, UserYearKeyWritable, AveWritable>{
	public class YearSumMapper extends Mapper<LongWritable, Text, IntWritable, IntWritable>{
	  2
	@Override
	protected void map(LongWritable key, Text value,
```

```
                Mapper<LongWritable, Text, IntWritable, IntWritable>.Context context)
                throws IOException, InterruptedException {
        // TODO Auto-generated method stub
        String[] tokens = value.toString().split("\\t");
        int userID = Integer.parseInt(tokens[0]);
        int year = Integer.parseInt(tokens[2]);

        context.write(new IntWritable(year), new IntWritable(1));
    }
}
```

YEARSUMREDUCER.JAVA

```
package com.bigdata.yearSum;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class YearSumReducer extends Reducer<IntWritable, IntWritable, IntWritable, IntWritable>{

    @Override
    protected void reduce(IntWritable key, Iterable<IntWritable> values,
                Reducer<IntWritable, IntWritable, IntWritable, IntWritable>.Context context)
                throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable v : values) {
            sum += v.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

PACKAGE COM.COMMONFRIENDS

COMMONAPP.JAVA

```
package com.commonFirends;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import com.concatenateFriends.ConcatenateApp;
import com.concatenateFriends.PersonFriendsMapper;
```

```
import com.concatenateFriends.PersonFriendsReducer;

public class CommonApp {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
	Configuration conf = new Configuration();
	Job job1 = Job.getInstance(conf, "Concatenate Person & Friends");

	//set driver class
	job1.setJarByClass(ConcatenateApp.class);

	//set mapper output
	job1.setMapOutputKeyClass(Text.class);
	job1.setMapOutputValueClass(Text.class);


	//set input output
	job1.setInputFormatClass(TextInputFormat.class);
	job1.setOutputFormatClass(TextOutputFormat.class);

	//output key and value
	job1.setOutputKeyClass(Text.class);
	job1.setOutputValueClass(Text.class);

	//set map reduce
	job1.setMapperClass(PersonFriendsMapper.class);
	job1.setReducerClass(PersonFriendsReducer.class);

	Path input = new Path("/FinalProject/Data/user_friends.dat");
//	Path output = new Path(args[1]);
	Path output = new Path("/FinalProject/Data/TestCommonFriends/Concatenate");
//	Path input = new Path("/Users/lvyan/Downloads/Data/user_friends.dat");
//	Path output = new Path("/Users/lvyan/Downloads/Data/concatenate");
	FileInputFormat.addInputPath(job1, input);
	FileOutputFormat.setOutputPath(job1, output);

	FileSystem hdfs = FileSystem.get(conf);
		if(hdfs.exists(output)){
			hdfs.delete(output, true);
		}

	job1.waitForCompletion(true);


Job job2 = Job.getInstance(conf, "Common Friends");

	//set driver class
	job2.setJarByClass(CommonFriendsApp.class);

	//set mapper output
	job2.setMapOutputKeyClass(Text.class);
	job2.setMapOutputValueClass(Text.class);
```

```
		//set input output
		job2.setInputFormatClass(TextInputFormat.class);
		job2.setOutputFormatClass(TextOutputFormat.class);

		//output key and value
		job2.setOutputKeyClass(Text.class);
		job2.setOutputValueClass(Text.class);

		//set map reduce
		job2.setMapperClass(UserFriendMapper.class);
		job2.setReducerClass(UserFriendReducer.class);

//		Path input = new Path(args[0]);
		Path output2 = new Path("/FinalProject/Data/TestCommonFriends/CommonFirneds");
		Path input2 = new Path("/FinalProject/Data/TestCommonFriends/Concatenate/part-*");
//		Path input2 = new Path("/Users/lvyan/Downloads/Data/Concatenate/part-*");
//		Path output2 = new Path("/Users/lvyan/Downloads/Data/commonFirends");
		FileInputFormat.addInputPath(job2, input2);
		FileOutputFormat.setOutputPath(job2, output2);

			if(hdfs.exists(output2)){
				hdfs.delete(output2, true);
			}

		job2.waitForCompletion(true);
	}

}
```

COMMONFRIENDSAPP.JAVA

```
package com.commonFirends;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import com.concatenateFriends.PersonFriendsMapper;
import com.concatenateFriends.PersonFriendsReducer;
public class CommonFriendsApp {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//	System.out.println( "Hello World!" );
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Common Friends");
```

```
		//set driver class
		job.setJarByClass(CommonFriendsApp.class);

		//set mapper output
		job.setMapOutputKeyClass(Text.class);
		job.setMapOutputValueClass(Text.class);


		//set input output
		job.setInputFormatClass(TextInputFormat.class);
		job.setOutputFormatClass(TextOutputFormat.class);

		//output key and value
		job.setOutputKeyClass(Text.class);
		job.setOutputValueClass(Text.class);

		//set map reduce
		job.setMapperClass(UserFriendMapper.class);
		job.setReducerClass(UserFriendReducer.class);

//		Path input = new Path(args[0]);
//		Path output = new Path(args[1]);
		Path input = new Path("/Users/lvyan/Downloads/Data/concatenate/common.dat");
		Path output = new Path("/Users/lvyan/Downloads/Data/commonFirends");

		FileInputFormat.addInputPath(job, input);
		FileOutputFormat.setOutputPath(job, output);

		FileSystem hdfs = FileSystem.get(conf);
			if(hdfs.exists(output)){
				hdfs.delete(output, true);
			}

		job.waitForCompletion(true);
	}
}
```

USERFRIENDMAPPER.JAVA

```
package com.commonFirends;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class UserFriendMapper extends Mapper<LongWritable, Text, Text, Text>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context
context)
				throws IOException, InterruptedException {
		// TODO Auto-generated method stub
```

```
//                  per.map(key, value, context);
                    String[] tokens = value.toString().split("\\t");
                    int userID = Integer.parseInt(tokens[0]);
                    String[] friends = tokens[1].split(",");
                    Text outKey;
                    for(String friend : friends) {
                            int friendID = Integer.parseInt(friend);
                            outKey = getOutKey(userID, friendID);
                            if(outKey == null) {
                                    continue;
                            }
                            context.write(outKey, new Text(tokens[1]));
                    }
            }

            private Text getOutKey(int userID, int friendID) {
                    // TODO Auto-generated method stub
                    if(userID == friendID) {
                            return null;
                    }
                    String out = "";
                    if(userID < friendID) {
                            out = userID + "," + friendID;
                    }else {
                            out = friendID + "," + userID;
                    }
                    return new Text(out);
            }
}
```

USERFRIENDREDUCER.JAVA

```
package com.commonFirends;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;
import java.util.TreeSet;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class UserFriendReducer extends Reducer<Text, Text, Text, Text>{
        @Override
        protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context)
                        throws IOException, InterruptedException {
                // TODO Auto-generated method stub
//              super.reduce(arg0, arg1, arg2);
                List<Set<Integer>> listOfValue = new ArrayList<Set<Integer>>();
                for(Text value : values) {
                        Set<Integer> set = new HashSet<Integer>();
                        String[] friendList = value.toString().split(",");
```

```
                    for(String friend : friendList) {
                        set.add(Integer.parseInt(friend));
                    }
                    listOfValue.add(set);
                }
                Set<Integer> commonFirneds = listOfValue.get(0);
                for(int i = 1; i < listOfValue.size(); i++) {
                    commonFirneds = intersection(commonFirneds, listOfValue.get(i));
                }
                context.write(key, new Text(commonFirneds.toString()));
        }

        private Set<Integer> intersection(Set<Integer> friends1, Set<Integer> friends2) {
                // TODO Auto-generated method stub
                if(friends1 == null || friends1.isEmpty()) {
                    return null;
                }
                if(friends2 == null || friends2.isEmpty()) {
                    return null;
                }
                if(friends1.size() < friends2.size()) {
                    return intersect(friends1, friends2);
                }else {
                    return intersect(friends2, friends1);
                }
        }

        private Set<Integer> intersect(Set<Integer> smallSet, Set<Integer> largeSet) {
                // TODO Auto-generated method stub
                Set<Integer> result = new TreeSet<Integer>();
                for(Integer x : smallSet) {
                    if(largeSet.contains(x)) {
                        result.add(x);
                    }
                }
                return result;
        }


}
```

PACKAGE COM.CONCATNATEFRIENDS

CONCATENATEAPP.JAVA

```
package com.concatenateFriends;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class ConcatenateApp {
        public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//      System.out.println( "Hello World!" );
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Concatenate Person & Friends");

        //set driver class
        job.setJarByClass(ConcatenateApp.class);

        //set mapper output
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);


        //set input output
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //output key and value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        //set map reduce
        job.setMapperClass(PersonFriendsMapper.class);
        job.setReducerClass(PersonFriendsReducer.class);

//      Path input = new Path(args[0]);
//      Path output = new Path(args[1]);
        Path input = new Path("/Users/lvyan/Downloads/Data/user_friends.dat");
        Path output = new Path("/Users/lvyan/Downloads/Data/concatenate");

        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);

        FileSystem hdfs = FileSystem.get(conf);
                if(hdfs.exists(output)){
                        hdfs.delete(output, true);
                }

        job.waitForCompletion(true);
 }
}
```

PERSONFRIENDSMAPPER.JAVA

```
package com.concatenateFriends;
import java.io.IOException;
```

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class PersonFriendsMapper extends Mapper<LongWritable, Text, Text, Text>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context
context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.map(key, value, context);
		if(key.get() == 0) {
			return;
		}
		String[] tokens = value.toString().split("\\t");
		String userID = tokens[0];
		String friendID = tokens[1];
		context.write(new Text(userID), new Text(friendID));

	}

}
```

PERSONFRIENDSREDUCER.JAVA

```
package com.concatenateFriends;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class PersonFriendsReducer extends Reducer<Text, Text, Text, Text>{

	@Override
	protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.reduce(arg0, arg1, arg2);
		String outvalue = "";
		for(Text value : values) {
			outvalue += value.toString() + ",";
		}
		outvalue = outvalue.substring(0, outvalue.length() - 1);
		context.write(key, new Text(outvalue));
	}

}
```

PACKAGE COM.RECOMMENDATION

ARTISTRATINGSUMWRITABLE.JAVA

```
package com.recommendation;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;
public class ArtistRatingSumWritable implements Writable{
	private int artistID;
	private int rating;
	private int sum;

	public ArtistRatingSumWritable() {
		super();
	}

	public ArtistRatingSumWritable(int artisID, int rating, int sum) {
		super();
		this.artistID = artisID;
		this.rating = rating;
		this.sum = sum;
	}

	public int getArtistID() {
		return artistID;
	}

	public void setArtistID(int artistID) {
		this.artistID = artistID;
	}

	public int getRating() {
		return rating;
	}

	public void setRating(int rating) {
		this.rating = rating;
	}

	public int getSum() {
		return sum;
	}

	public void setSum(int sum) {
		this.sum = sum;
	}
	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		artistID = input.readInt();
		rating = input.readInt();
		sum = input.readInt();
	}
```

```
	5
	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeInt(artistID);
		output.writeInt(rating);
		output.writeInt(sum);
	}

	public String toString() {
		return this.getArtistID() + "\t" + this.getRating() + "\t" + this.getSum();
	}

}
```

CORRELATIONWRITABLE.JAVA

```
package com.recommendation;
1
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.Writable;

public class CorrelationWritable implements Writable{

	private double pearson;
	private double cosine;
	private double jaccard;

	public CorrelationWritable(){
		super();
	}

	public CorrelationWritable(double i, double j, double k) {
		super();
		this.pearson = i;
		this.cosine = j;
		this.jaccard = k;
	}
	1
	public double getPearson() {
		return pearson;
	}

	public void setPearson(double pearson) {
		this.pearson = pearson;
	}

	public double getCosine() {
		return cosine;
	}

	public void setCosine(double cosine) {
```

```
		this.cosine = cosine;
	}
	public double getJaccard1() {
		return jaccard;
	}

	public void setJaccard1(double jaccard1) {
		this.jaccard = jaccard1;
	}

	public void readFields(DataInput input) throws IOException {
		// TODO Auto-generated method stub
		pearson = input.readDouble();
		cosine = input.readDouble();
		jaccard = input.readDouble();
	}

	public void write(DataOutput output) throws IOException {
		// TODO Auto-generated method stub
		output.writeDouble(pearson);
		output.writeDouble(cosine);
		output.writeDouble(jaccard);
	}
	public String toString() {
		return this.getPearson() + "\t" + this.getCosine() + "\t" + this.getJaccard1();
	}
}
```

MRPHASE1APP.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MRPhase1App {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//	System.out.println( "Hello World!" );
	Configuration conf = new Configuration();
	Job job = Job.getInstance(conf, "Recommendation Phase 1");

	//set driver class
	job.setJarByClass(MRPhase1App.class);
```

```
		//set mapper output
		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(UserRatingSumWritable.class);


		//set input output
		job.setInputFormatClass(TextInputFormat.class);
		job.setOutputFormatClass(TextOutputFormat.class);

		//output key and value
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(ArtistRatingSumWritable.class);

		//set map reduce
		job.setMapperClass(UserRatingMapper.class);
		job.setReducerClass(UserRatingReducer.class);

		Path input = new Path("/FinalProject/Data/user_artists.dat");
		Path output = new Path("/FinalProject/Data/TestRecommondation/phase1");
//		Path input = new Path("/Users/Ivyan/Downloads/Data/user_artists.dat");
//		Path output = new Path("/Users/Ivyan/Downloads/Data/recommondation/phase1");
//		1
		FileInputFormat.addInputPath(job, input);
		FileOutputFormat.setOutputPath(job, output);

		FileSystem hdfs = FileSystem.get(conf);
			if(hdfs.exists(output)){
				hdfs.delete(output, true);
			}

		job.waitForCompletion(true);
	}
}
```

MRPHASE2APP.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MRPhase2App {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
```

```
//    System.out.println( "Hello World!" );
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Recommendation Phase 2");

        //set driver class
        job.setJarByClass(MRPhase2App.class);

        //set mapper output
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(ArtistRatingSumWritable.class);


        //set input output
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        //output key and value
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        //set map reduce
        job.setMapperClass(Phase2Mapper.class);
        job.setReducerClass(Phase2Reducer.class);

        Path input = new Path("/FinalProject/Data/TestRecommondation/phase1/part-r-*");
        Path output = new Path("/FinalProject/Data/TestRecommondation/phase2");
//      Path input = new Path("/Users/lvyan/Downloads/Data/recommondation/phase1/part-r-*");
//      Path output = new Path("/Users/lvyan/Downloads/Data/recommondation/phase2");
        FileInputFormat.addInputPath(job, input);
        FileOutputFormat.setOutputPath(job, output);

        FileSystem hdfs = FileSystem.get(conf);
                if(hdfs.exists(output)){
                        hdfs.delete(output, true);
                }

        job.waitForCompletion(true);
 }
}
```

MRPHASE3APP.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MRPhase3App {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//	System.out.println( "Hello World!" );
	Configuration conf = new Configuration();
	Job job = Job.getInstance(conf, "Recommendation Phase 3");

	//set driver class
	job.setJarByClass(MRPhase3App.class);

	//set mapper output
	job.setMapOutputKeyClass(Text.class);
	job.setMapOutputValueClass(Text.class);

	//set input output
	job.setInputFormatClass(TextInputFormat.class);
	job.setOutputFormatClass(TextOutputFormat.class);

	//output key and value
	job.setOutputKeyClass(Text.class);
	job.setOutputValueClass(CorrelationWritable.class);

	//set map reduce
	job.setMapperClass(Phase3Mapper.class);
	job.setReducerClass(Phase3Reducer.class);

	Path input = new Path("/FinalProject/Data/TestRecommondation/phase2/part-r-*");
	Path output = new Path("/FinalProject/Data/TestRecommondation/phase3");
//	Path input = new Path("/Users/lvyan/Downloads/Data/recommondation/phase2/part-r-*");
//	Path output = new Path("/Users/lvyan/Downloads/Data/recommondation/phase3");
	FileInputFormat.addInputPath(job, input);
	FileOutputFormat.setOutputPath(job, output);

	FileSystem hdfs = FileSystem.get(conf);
		if(hdfs.exists(output)){
			hdfs.delete(output, true);
		}

	job.waitForCompletion(true);
 }
}
```

MRPHASE4APP.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MRPhase4App {
	public static void main( String[] args ) throws IOException, ClassNotFoundException,
InterruptedException{
//	System.out.println( "Hello World!" );
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Recommendation Phase 4");

		//set driver class
		job.setJarByClass(MRPhase4App.class);

		//set mapper output
		job.setMapOutputKeyClass(IntWritable.class);
		job.setMapOutputValueClass(IntWritable.class);

		//set input output
		job.setInputFormatClass(TextInputFormat.class);
		job.setOutputFormatClass(TextOutputFormat.class);

		//output key and value
		job.setOutputKeyClass(IntWritable.class);
		job.setOutputValueClass(Text.class);

		//set map reduce
		job.setMapperClass(Phase4Mapper.class);
		job.setReducerClass(Phase4Reducer.class);

		Path input = new Path("/FinalProject/Data/TestRecommondation/phase3/part-r-*");
		Path output = new Path("/FinalProject/Data/TestRecommondation/phase4");
//		Path input = new Path("/Users/lvyan/Downloads/Data/recommondation/phase3/part-r-*");
//		Path output = new Path("/Users/lvyan/Downloads/Data/recommondation/phase4");

		FileInputFormat.addInputPath(job, input);
		FileOutputFormat.setOutputPath(job, output);

		FileSystem hdfs = FileSystem.get(conf);
			if(hdfs.exists(output)){
				hdfs.delete(output, true);
			}

		job.waitForCompletion(true);
  }
}
```

PHASE2MAPPER .JAVA

```
package com.recommendation;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Phase2Mapper  extends Mapper<LongWritable, Text, IntWritable, ArtistRatingSumWritable>{

	@Override
	protected void map(LongWritable key, Text value,
			Mapper<LongWritable, Text, IntWritable, ArtistRatingSumWritable>.Context context)
			throws IOException, InterruptedException {
		String[] tokens = value.toString().split("\\t");
		IntWritable outKey = new IntWritable(Integer.parseInt(tokens[0]));
		int artistID = Integer.parseInt(tokens[1]);
		int rating = Integer.parseInt(tokens[2]);
		int numberOfRaters = Integer.parseInt(tokens[3]);
		ArtistRatingSumWritable outValue = new ArtistRatingSumWritable(artistID, rating,
numberOfRaters);
//		System.out.println(artistID);
		context.write(outKey, outValue);
	}
}
```

PHASE2REDUCER.JAVA

```
package com.recommendation;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Phase2Reducer extends Reducer<IntWritable, ArtistRatingSumWritable, Text, Text>{

	@Override
	protected void reduce(IntWritable key, Iterable<ArtistRatingSumWritable> values,
			Reducer<IntWritable, ArtistRatingSumWritable, Text, Text>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.reduce(arg0, arg1, arg2);
		//使输入的 value 两两组合起来
		List<ArtistRatingSumWritable> valueList = new ArrayList<ArtistRatingSumWritable>();
		for(ArtistRatingSumWritable ars : values) {
			valueList.add(new ArtistRatingSumWritable(ars.getArtistID(), ars.getRating(),
ars.getSum()));
		}

		List<List<ArtistRatingSumWritable>> list = generateUniqueCombinations(valueList);
```

```
		for(List<ArtistRatingSumWritable> pair : list) {
			ArtistRatingSumWritable a1 = pair.get(0);
			ArtistRatingSumWritable a2 = pair.get(1);
			Text reducerKey = new Text(a1.getArtistID() + "," + a2.getArtistID());
			int ratingProduct = a1.getRating() * a2.getRating();
			int rating1Squared = a1.getRating() * a1.getRating();
			int rating2Squared = a2.getRating() * a2.getRating();
			String out = a1.getRating() + "," + a1.getSum() + "," + a2.getRating() + "," +
					a2.getSum() + "," + ratingProduct + "," + rating1Squared + "," +
rating2Squared;
			Text reducerValue = new Text(out);
			context.write(reducerKey, reducerValue);
		}
	}

	private List<List<ArtistRatingSumWritable>> generateUniqueCombinations(List<ArtistRatingSumWritable>
valueList) {
		// TODO Auto-generated method stub
		List<List<ArtistRatingSumWritable>> list = new ArrayList<List<ArtistRatingSumWritable>>();
		for(int i = 0; i < valueList.size() - 1; i++) {
			ArtistRatingSumWritable v1 = valueList.get(i);
			ArtistRatingSumWritable v2 = valueList.get(i + 1);
			if(v1.getArtistID() < v2.getArtistID()) {
//				System.out.println("++++++++++++++++"+valueList.size());
				List<ArtistRatingSumWritable> pair = new
ArrayList<ArtistRatingSumWritable>();
				pair.add(v1);
				pair.add(v2);
				list.add(pair);
			}
		}
		return list;
	}
}
```

PHASE3MAPPER.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class Phase3Mapper extends Mapper<LongWritable, Text, Text, Text>{

	@Override
	protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, Text>.Context
context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.map(key, value, context);
		String[] tokens = value.toString().split("\\t");
		Text outKey = new Text(tokens[0]);
```

```
                Text outValue = new Text(tokens[1]);
                context.write(outKey, outValue);
        }
}
```

PHASE3REDUCER.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class Phase3Reducer extends Reducer<Text, Text, Text, CorrelationWritable> {
        @Override
        protected void reduce(Text key, Iterable<Text> values,
                        Reducer<Text, Text, Text, CorrelationWritable>.Context context) throws IOException,
InterruptedException {
                // TODO Auto-generated method stub
//              super.reduce(arg0, arg1, arg2);
                int groupSize = 0;
                int dotProduct = 0;
                int rating1Sum = 0;
                int rating2Sum = 0;
                int rating1NormSq = 0; // sum of rating1Squared
                int rating2NormSq = 0;
                int maxNumOfumRaters1 = 0; // max of numOfRaters1
                int maxNumOfumRaters2 = 0;

                for (Text value : values) {
                        groupSize++;
                        // parse the value
                        String[] tokens = value.toString().split(",");
                        int rating1 = Integer.parseInt(tokens[0]);
                        int numOfRaters1 = Integer.parseInt(tokens[1]);
                        int rating2 = Integer.parseInt(tokens[2]);
                        int numOfRaters2 = Integer.parseInt(tokens[3]);
                        int ratingProduct = Integer.parseInt(tokens[4]);
                        int rating1Squared = Integer.parseInt(tokens[5]);
                        int rating2Squared = Integer.parseInt(tokens[6]);

                        dotProduct += ratingProduct;
                        rating1Sum += rating1;
                        rating2Sum += rating2;
                        rating1NormSq += rating1Squared;
                        rating2NormSq += rating2Squared;

                        if (numOfRaters1 > maxNumOfumRaters1) {
                                maxNumOfumRaters1 = numOfRaters1;
                        }

                        if (numOfRaters2 > maxNumOfumRaters2) {
                                maxNumOfumRaters2 = numOfRaters2;
                        }
                }
```

```
                double pearson = calculatePearsonCorrelation(groupSize, dotProduct, rating1Sum, rating2Sum,
rating1NormSq,
                                rating2NormSq);

                double cosine = calculateCosineCorrelation(dotProduct, Math.sqrt(rating1NormSq),
Math.sqrt(rating2NormSq));

                double jaccard = calculateJaccardCorrelation(groupSize, maxNumOfumRaters1,
maxNumOfumRaters2);

                CorrelationWritable outValue = new CorrelationWritable(pearson, cosine, jaccard);

                context.write(key, outValue);
        }

        static double calculatePearsonCorrelation(double size, double dotProduct, double rating1Sum, double
rating2Sum,
                        double rating1NormSq, double rating2NormSq) {
                double numerator = size * dotProduct - rating1Sum * rating2Sum;
                double denominator = Math.sqrt(size * rating1NormSq - rating1Sum * rating1Sum)
                                * Math.sqrt(size * rating2NormSq - rating2Sum * rating2Sum);
                return numerator / denominator;
        }

        static double calculateCosineCorrelation(double dotProduct, double rating1Norm, double rating2Norm) {
                return dotProduct / (rating1Norm * rating2Norm);
        }

        static double calculateJaccardCorrelation(double inCommon, double totalA, double totalB) {
                double union = totalA + totalB - inCommon;
                return inCommon / union;
        }
}
```

PHASE4MAPPER.JAVA

```
package com.recommendation;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Phase4Mapper extends Mapper<LongWritable, Text, IntWritable, IntWritable>{

        @Override
        protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, IntWritable,
IntWritable>.Context context)
                        throws IOException, InterruptedException {
                // TODO Auto-generated method stub
```

```
//                  per.map(key, value, context);
                    String[] tokens = value.toString().split("\\t");
                    String[] artists = tokens[0].split(",");
                    double p = Double.parseDouble(tokens[1]);
                    double c = Double.parseDouble(tokens[2]);
                    double j = Double.parseDouble(tokens[3]);
                    int a1 = Integer.parseInt(artists[0]);
                    int a2 = Integer.parseInt(artists[1]);
                    if((p >= 0.5 || c <= 0.5 || j >= 0.5) && a1 != a2) {
                            context.write(new IntWritable(a1), new IntWritable(a2));
                            context.write(new IntWritable(a2), new IntWritable(a1));
                    }
            }

}
```

PHASE4REDUCER.JAVA

```
package com.recommendation;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Phase4Reducer extends Reducer<IntWritable, IntWritable, IntWritable, Text>{

        @Override
        protected void reduce(IntWritable key, Iterable<IntWritable> values,
                        Reducer<IntWritable, IntWritable, IntWritable, Text>.Context context)
                        throws IOException, InterruptedException {
                // TODO Auto-generated method stub
//              super.reduce(arg0, arg1, arg2);
                String out = "";
                for(IntWritable t : values) {
                        out += t.get() + ",";
                }
                out = out.substring(0, out.length() - 1);
                context.write(key, new Text(out));
        }

}
```

USERRATINGMAPPER.JAVA

```
package com.recommendation;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
public class UserRatingMapper extends Mapper<LongWritable, Text, IntWritable, UserRatingSumWritable>{

	@Override
	protected void map(LongWritable key, Text value,
			Mapper<LongWritable, Text, IntWritable, UserRatingSumWritable>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.map(key, value, context);
		if(key.get() == 0) {
			return;
		}
		String[] tokens = value.toString().split("\\t");
		int userID = Integer.parseInt(tokens[0]);
		int artistID = Integer.parseInt(tokens[1]);
		int rating = Integer.parseInt(tokens[2]);
		IntWritable outKey = new IntWritable(artistID);
		UserRatingSumWritable outValue = new UserRatingSumWritable(userID, rating);
		context.write(outKey, outValue);
	}

}
```

USERRATINGREDUCER.JAVA

```
package com.recommendation;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.Reducer;

public class UserRatingReducer extends Reducer<IntWritable, UserRatingSumWritable, IntWritable,
ArtistRatingSumWritable>{

	@Override
	protected void reduce(IntWritable key, Iterable<UserRatingSumWritable> values,
			Reducer<IntWritable, UserRatingSumWritable, IntWritable,
ArtistRatingSumWritable>.Context context)
			throws IOException, InterruptedException {
		// TODO Auto-generated method stub
//		super.reduce(arg0, arg1, arg2);
		int sum = 0;
		List<ArtistRatingSumWritable> list = new ArrayList<ArtistRatingSumWritable>();
		for(UserRatingSumWritable ur : values) {
			sum++;
//			sum += ur.getRating();
			int artistID = key.get();
			int rating = ur.getRating();
			ArtistRatingSumWritable outValue = new ArtistRatingSumWritable(artistID, rating,
ur.getUserID());
			list.add(outValue);
		}
//		IntWritable outKey = new IntWritable(ur.getUserID());
```

```
                for(ArtistRatingSumWritable v : list) {
                        IntWritable outKey = new IntWritable(v.getSum());
                        v.setSum(sum);
                        context.write(outKey, v);
                }
        }
}
```

USERRATINGSUMWRITABLE.JAVA

```
package com.recommendation;
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import org.apache.hadoop.io.Writable;

public class UserRatingSumWritable implements Writable{

        private int userID;
        private int rating;

        public UserRatingSumWritable() {
                super();
        }

        public UserRatingSumWritable(int userID, int rating) {
                super();
                this.userID = userID;
                this.rating = rating;
        }

                16
        public int getUserID() {
                return userID;
        }


        public void setUserID(int userID) {
                this.userID = userID;
        }


        public int getRating() {
                return rating;
        }


        public void setRating(int rating) {
                this.rating = rating;
        }

                5
        public void readFields(DataInput input) throws IOException {
```

```
                // TODO Auto-generated method stub
                userID = input.readInt();
                rating = input.readInt();
        }
        public void write(DataOutput output) throws IOException {
                // TODO Auto-generated method stub
                output.writeInt(userID);
                output.writeInt(rating);
        }
        public String toString() {
                return this.getUserID() + "\t" + this.getRating();
        }
}
```

POM.XML

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.bigdata</groupId>
 <artifactId>BigDataFinalProject</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <name>BigDataFinalProject</name>
 <url>http://maven.apache.org</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 </properties>

 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core -->
                <dependency>
                        <groupId>org.apache.hadoop</groupId>
                        <artifactId>hadoop-core</artifactId>
                        <version>1.2.1</version>
                </dependency>
                <dependency>
                        <groupId>org.apache.mahout</groupId>
                        <artifactId>mahout-mr</artifactId>
                        <version>0.13.0</version>
```

```
                </dependency>
                <dependency>
                        <groupId>org.apache.mahout</groupId>
                        <artifactId>mahout-core</artifactId>
                        <version>0.9</version>
                </dependency>
                <dependency>
                        <groupId>org.mapdb</groupId>
                        <artifactId>mapdb</artifactId>
                        <version>3.0.7</version>
                </dependency>
 </dependencies>
</project>
```

HIVE CODE

```
with temp_year as(
select distinct userid, artistid, year
from user_time),
temp as(
select ut.year, ua.artistid,
sum(ua.weight) as totalTimes,
rank() over(partition by ut.year order by sum(ua.weight) desc) as rk
from  user_artist ua
join temp_year ut
on ua.userid = ut.userid and ua.artistid = ut.artistid
group by ua.artistid, ut.year
)
select year, artistid, totalTimes
from temp
where rk < 11
order by year, rk
```

PIG CODE

```
user_time = load '/Users/lvyan/Downloads/Data/user_taggedartists.dat' using PigStorage('\t') as (userid, artistid,
tagid, day, month, year);
user_time = foreach user_time generate userid, artistid, year;
user_time = distinct user_time;
user_artist = load '/Users/lvyan/Downloads/Data/user_artists.dat' using PigStorage('\t') as (userid, artistid,
weight:int);
user_weight_year = join user_artist by (userid, artistid), user_time by (userid, artistid);
user_weight_year = foreach user_weight_year generate year, user_artist::artistid, weight;

data_1956 = filter user_weight_year by year == 1956;
group_1956 = group data_1956 by artistid;
sum_1956 = foreach group_1956 generate $0, SUM($1.weight);
order_1956 = order sum_1956 by $1 desc;
top_1956 = limit order_1956 10;

data_1957 = filter user_weight_year by year == 1957;
group_1957 = group data_1957 by artistid;
sum_1957 = foreach group_1957 generate $0, SUM($1.weight);
```

```
order_1957 = order sum_1957 by $1 desc;
top_1957 = limit order_1957 10;

data_2005 = filter user_weight_year by year == 2005;
group_2005 = group data_2005 by artistid;
sum_2005 = foreach group_2005 generate $0, SUM($1.weight);
order_2005 = order sum_2005 by $1 desc;
top_2005 = limit order_2005 10;

data_2006 = filter user_weight_year by year == 2006;
group_2006 = group data_2006 by artistid;
sum_2006 = foreach group_2006 generate $0, SUM($1.weight);
order_2006 = order sum_2006 by $1 desc;
top_2006 = limit order_2006 10;

data_2007 = filter user_weight_year by year == 2007;
group_2007 = group data_2007 by artistid;
sum_2007 = foreach group_2007 generate $0, SUM($1.weight);
order_2007 = order sum_2007 by $1 desc;
top_2007 = limit order_2007 10;

data_2008 = filter user_weight_year by year == 2008;
group_2008 = group data_2008 by artistid;
sum_2008 = foreach group_2008 generate $0, SUM($1.weight);
order_2008 = order sum_2008 by $1 desc;
top_2008 = limit order_2008 10;

data_2009 = filter user_weight_year by year == 2009;
group_2009 = group data_2009 by artistid;
sum_2009 = foreach group_2009 generate $0, SUM($1.weight);
order_2009 = order sum_2009 by $1 desc;
top_2009 = limit order_2009 10;

data_2010 = filter user_weight_year by year == 2010;
group_2010 = group data_2010 by artistid;
sum_2010 = foreach group_2010 generate $0, SUM($1.weight);
order_2010 = order sum_2010 by $1 desc;
top_2010 = limit order_2010 10;

data_2011 = filter user_weight_year by year == 2011;
group_2011 = group data_2011 by artistid;
sum_2011 = foreach group_2011 generate $0, SUM($1.weight);
order_2011 = order sum_2011 by $1 desc;
top_2011 = limit order_2011 10;

data_1976 = filter user_weight_year by year == 1976;
group_1976 = group data_1976 by artistid;
sum_1976 = foreach group_1976 generate $0, SUM($1.weight);
order_1976 = order sum_1976 by $1 desc;
top_1976 = limit order_1976 10;

Store top_1976 into '/Users/lvyan/Downloads/Data/Top10ByPig/2' using PigStorage();
Store top_1956 into '/Users/lvyan/Downloads/Data/Top10ByPig/0' using PigStorage();
```

```
Store top_1957 into '/Users/lvyan/Downloads/Data/Top10ByPig/1' using PigStorage();
Store top_2005 into '/Users/lvyan/Downloads/Data/Top10ByPig/3' using PigStorage();
Store top_2006 into '/Users/lvyan/Downloads/Data/Top10ByPig/4' using PigStorage();
Store top_2007 into '/Users/lvyan/Downloads/Data/Top10ByPig/5' using PigStorage();
Store top_2008 into '/Users/lvyan/Downloads/Data/Top10ByPig/6' using PigStorage();
Store top_2009 into '/Users/lvyan/Downloads/Data/Top10ByPig/7' using PigStorage();
Store top_2010 into '/Users/lvyan/Downloads/Data/Top10ByPig/8' using PigStorage();
Store top_2011 into '/Users/lvyan/Downloads/Data/Top10ByPig/9' using PigStorage();
```

Final project_Yan Lyu

ORIGINALITY REPORT

32%	25%	21%	28%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Northeastern University Student Paper	16%
2	bigdataprogrammers.com Internet Source	2%
3	blog.ring.idv.tw Internet Source	1%
4	www.javafun.cn Internet Source	1%
5	geek521.com Internet Source	1%
6	Sameer Wadkar, Madhu Siddalingaiah. "Pro Apache Hadoop", Springer Nature, 2014 Publication	1%
7	Submitted to University of Central Florida Student Paper	1%
8	www.buyukveri.co Internet Source	1%
9	in7in.net	

	Internet Source	1%
10	www.ziu.me Internet Source	<1%
11	puremonkey2010.blogspot.com Internet Source	<1%
12	www.deploymentzone.com Internet Source	<1%
13	www.knime.org Internet Source	<1%
14	Submitted to University of Liverpool Student Paper	<1%
15	blog.fens.me Internet Source	<1%
16	Gary Mak, Josh Long, Daniel Rubio. "Spring Recipes", Springer Nature, 2010 Publication	<1%
17	Perry Xiao. "Practical Java® Programming for IoT, AI, and Blockchain", Wiley, 2018 Publication	<1%
18	Submitted to Queen Mary and Westfield College Student Paper	<1%
19	Submitted to University College London Student Paper	<1%

20	sujitpal.blogspot.com Internet Source	<1%
21	K.G. Srinivasa, Anil Kumar Muppalla. "Guide to High Performance Distributed Computing", Springer Nature, 2015 Publication	<1%
22	Submitted to Wright State University Student Paper	<1%
23	www.tech126.com Internet Source	<1%
24	Jeff Friesen. "Learn Java for Android Development", Springer Nature, 2013 Publication	<1%
25	Submitted to CSU, Dominguez Hills Student Paper	<1%
26	Johannes Moe. "Integrated Design of Tanker Structures", Elsevier BV, 1973 Publication	<1%
27	www.roman10.net Internet Source	<1%
28	Submitted to Arizona State University Student Paper	<1%
29	tech.e800.com.cn Internet Source	<1%

30	timepasstechies.com Internet Source	<1%
31	Submitted to The Hong Kong Polytechnic University Student Paper	<1%
32	henning.kropponline.de Internet Source	<1%
33	unmeshasreeveni.blogspot.in Internet Source	<1%
34	amac4.blogspot.com Internet Source	<1%
35	Submitted to UT, Dallas Student Paper	<1%
36	www.cfanz.cn Internet Source	<1%
37	Submitted to Istanbul Bilgi University Student Paper	<1%
38	free2code.net Internet Source	<1%
39	forum.hadoop.tw Internet Source	<1%
40	Submitted to Indus International School Student Paper	<1%

41	Deepak Vohra. "Practical Hadoop Ecosystem", Springer Nature, 2016 Publication	<1%
42	www.blogjava.net Internet Source	<1%
43	Jeremy Kerfs. "Beginning Android Tablet Games Programming", Springer Nature, 2011 Publication	<1%
44	Jeff Linwood, Dave Minter. "Building Portals with the Java Portlet API", Springer Nature, 2004 Publication	<1%
45	"Pro Java Programming", Springer Nature, 2005 Publication	<1%
46	"Distributed Computing and Artificial Intelligence, Special Sessions, 15th International Conference", Springer Science and Business Media LLC, 2019 Publication	<1%
47	S. Kannan, S. Karuppusamy, A. Nedunchezhian, P. Venkateshan, P. Wang, N. Bojja, A. Kejariwal. "Big Data Analytics for Social Media", Elsevier BV, 2016 Publication	<1%
48	Submitted to Maharishi University of Management Student Paper	<1%

49	www.cs.bgu.ac.il Internet Source	<1%
50	Pin-Yu Chen, Alfred O. Hero. "Deep Community Detection", IEEE Transactions on Signal Processing, 2015 Publication	<1%
51	Mark Lui, Mario Gray, Andy Chan, Josh Long. "Pro Spring Integration", Springer Nature, 2011 Publication	<1%
52	www.edwardkim.pe.kr Internet Source	<1%
53	Submitted to CSU, San Jose State University Student Paper	<1%
54	hadoop-examples.blogspot.com Internet Source	<1%
55	blog.triplez.cn Internet Source	<1%
56	www.philippeadjiman.com Internet Source	<1%
57	www.intermine.org Internet Source	<1%
58	Submitted to Indian Institute of Technology, Madras Student Paper	<1%

59	"XML for Bioinformatics", Springer Nature, 2005 Publication	<1%
60	Eric A. Nyamsi. "Projektmanagement mit Scrum", Springer Science and Business Media LLC, 2019 Publication	<1%
61	Submitted to Swinburne University of Technology Student Paper	<1%
62	Jeff Friesen. "Learn Java for Android Development", Springer Nature, 2014 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		