**INFO 7275 Movies Recommendation System using Hadoop**

1. **Problem statement**

The data set for this project is Netflix Prize Data Set. There are 480000 customers in this dataset, and each customer has a unique user id. Also, each customer has rated many movies. There are over 17,000 rating records in this data set. In the movie details data set, there are three columns in it, and each column is a feature of movies. They are movie id, movie name, and released year. This project is developed as a recommendation system. Because of this goal, we should use the prediction machine learning algorithm as the analysis algorithm. In this project, we use Item Collaborative Filtering to predict which movie a customer should be like depending on the rating records from this customer. After that, recommend these movies for this certain customer. Also, I use Join Design Pattern to combine the rating file and movie details file. Using the TopK algorithm, we can get top 100 highest rating movies from the combination file. Inverted Index method was also used in this project, customer not only can search a movie with the movie id, but also with a key word of movie title.

1. **Dataset Link:**

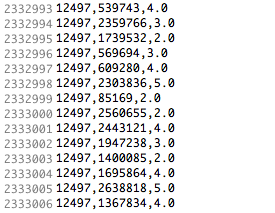
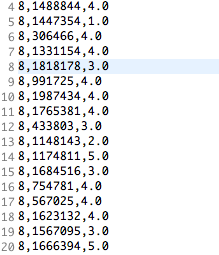
<http://courses.cs.washington.edu/courses/csep546/10sp/hw1/NDA.htm>

1. **Machine Learning Algorithms**
   1. **Item Collaborative Filtering (Item CF)**

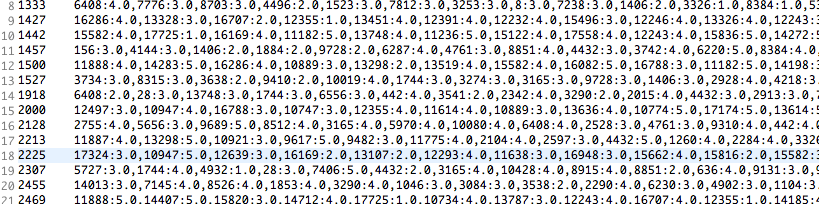
Using this algorithm, the recommend result comes from movies similarity based on each customer’s ratings for movies. After several transformation of data, we can get the movies similarity table. Also, we need join the movies similarity table and customer rating table. In this way, we can get the predicted results about how much score this customer will give to those movies he or she has not yet watched. Of course, the movie which get the highest score will be the most recommended movie for this curtain customer.

* + 1. **Pre-process Data**

Like the picture shown below, the raw movie\_rating data set has three columns. The first is Movie id, second is Customer id, and last is rating.



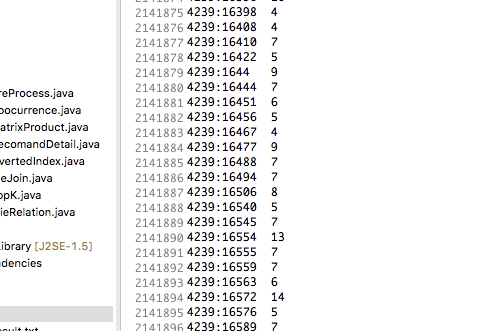
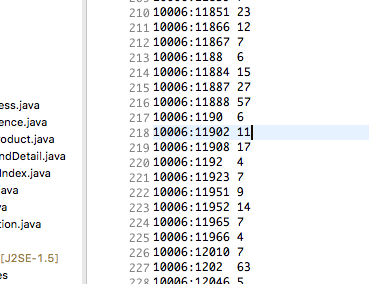
By using the A\_preprocess MapReduce method (shown in the appendix part), we can get a movie\_rating matrix, we call this Aresult.txt, and the data would be like:



Description : The key of this result is customer id. The value of it is a List. It contains all the movies this customer has already rated and the rating score for each movie. Each moive \_rating pair is seperated by a comma sign. And, movie and its rating seperated by a colon. This step is going to dividing customer by movies.

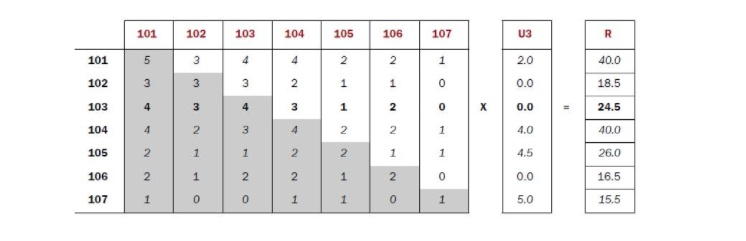
* + 1. **Products Co-ocurrance Matrix**

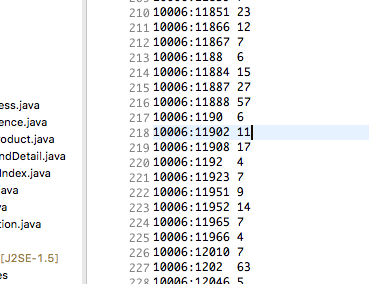
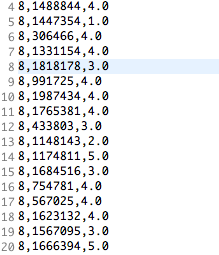
Depending on ratings of each movie, we get get a Co-ocurrance Matrix. This matrix stands for the the times that two movies are rated by same customer. The input file for this B\_Coocurrance MapReduce method (shown in appendix part) is the result of last step. The output is a matrix, whose key is the movie1 : movie2 pair, and the value is the times that these two movies rated by same customer.

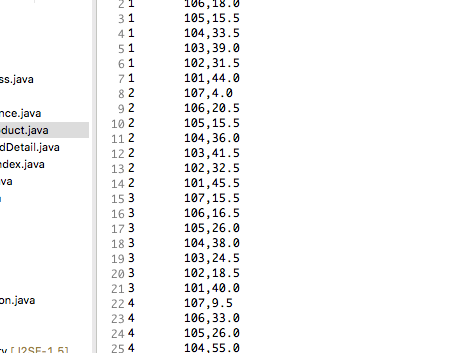
* + 1. **Co-ocurrance mutiply movie\_rating**

This step is a multiply process, in the othere word, it is a Full-Join process between two matrix. We combine the Co-ocurrance matrix and the movie\_rating matrix.



And the result of the C\_matrixProduct (shown in the appendix part) is the final predicted result.



Description: The key of this result is the customer id, and the value is a key-value pair. They are movie id and the rating this customer should give to this movie. For accurate caculating, we need nomalize the rating part. Because rating is the sum of all ratings for one certain movie, we need divide rating times of this movie.

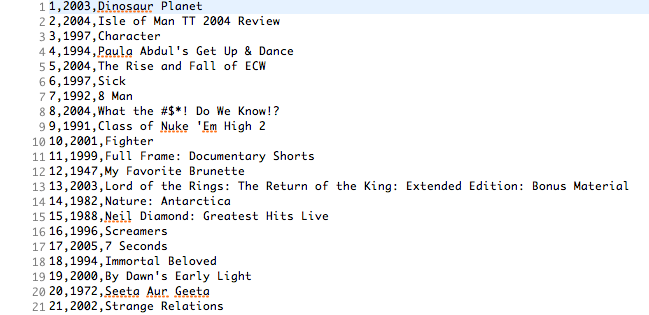
* + 1. **Filtering and Matching**

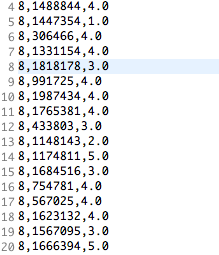
Last step’s result shows all movies and ratings for each customer including the movies this customer had already watched. Thus, we need filter the movie a customer has not yet watched and recommend the hight rating ones by using D\_RecommandDetail MapReduce method (shown in the appendix part).

And, for customer convieneince, we join the result data set and the movie\_titles data set to show recommanded movies and movie names at the same time.

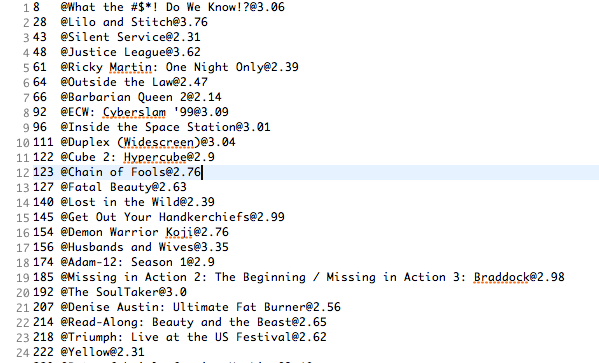
* 1. **Map Side Join**

In this method, we will join two data sets. One is the movie\_rating data set, another is movie\_title dataset.



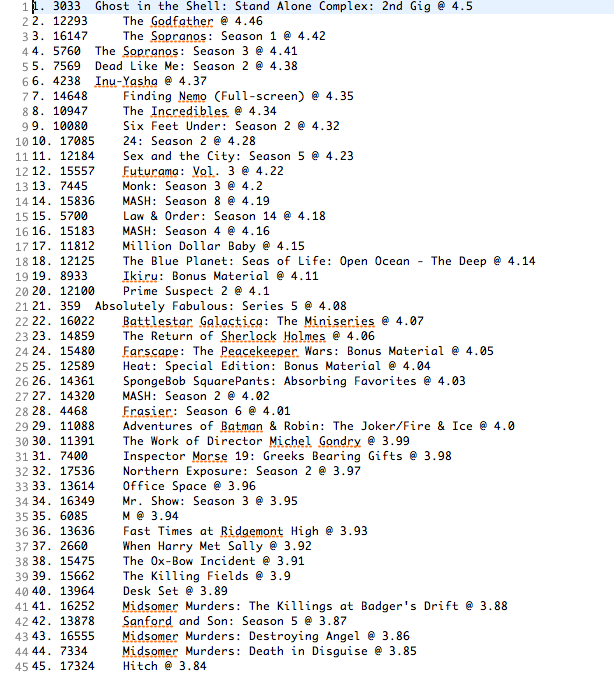


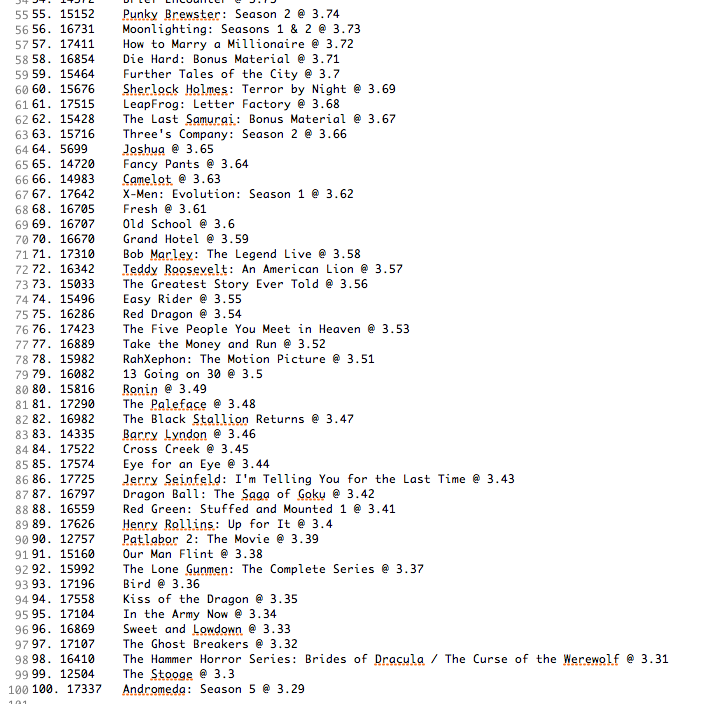
The result of F\_FileJoin MapReduce methd (shown in the appendix part) contains three columns. First is movie id, second is movie name, last one is average rating. These three features are separated by “@” sign.



* 1. **Top K Algorithm**

G\_TopK MapReduce method (shown in the appendix part) compare the average rating of each movie and provide the Top 100 movies and their name, average rating. input data set is the Map Side Join MapReduce method’s output.

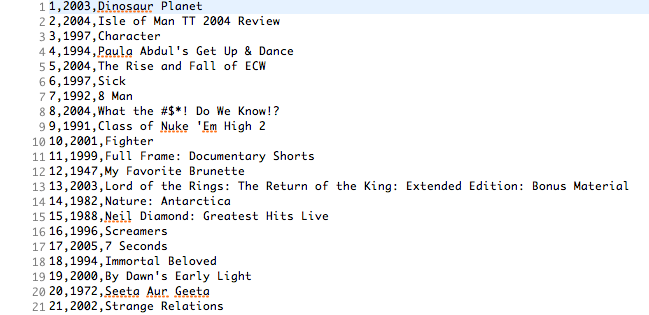




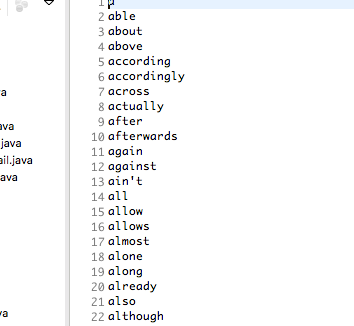
* 1. **Comparison : Inverted Index vs Tradition**

In the tradition way, if a customer wants to search a movie, the system will scan the whole data set to find that certain one. However, by using Inverted Index method, we can improve the system performance a lot. For example, search a movie by a key word, and the system will give you a List which contains all the books whose name contains the key word.

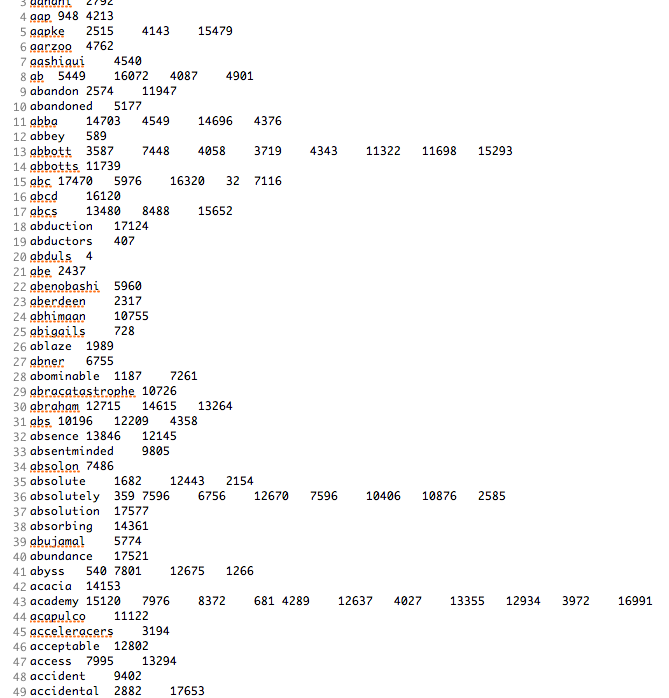
Input data set is the movie\_titles data set.



There are some non-meaningful words in the movie names, such as “a”, “be”. We call them “Stop Word”. We do not want to consider them, so we need skip them. StopWord data set is:



The output of E\_InvertedIndex MapReduce method (shown in the appendix part) has two parts. The key is key words, and the value is a List which contains movie ids.



For example, if a customer searched a key word “abraham”, system will provide a movie list of “12715, 14615, 13264”.

1. **Appendix**

**4.1 A\_preprocess.java**

**public** **class** A\_preProcess {

**public** **static** **class** myMapperA **extends** Mapper<LongWritable, Text, IntWritable, Text>{

@Override

**public** **void** map(LongWritable key, Text value, Context contex) **throws** IOException, InterruptedException {

String[] movie\_user\_rating = value.toString().trim().split(",");

**int** userId = Integer.*parseInt*(movie\_user\_rating[1]);

String movieId = movie\_user\_rating[0];

String rating = movie\_user\_rating[2];

contex.write(**new** IntWritable(userId), **new** Text(movieId + ":" + rating));

}

}

**public** **static** **class** myReducerA **extends** Reducer<IntWritable,Text, IntWritable, Text> {

// -> user1 movie1 : rating, movie2 : rating

@Override

**public** **void** reduce(IntWritable key, Iterable<Text> values,

Context context) **throws** IOException, InterruptedException {

StringBuilder sb = **new** StringBuilder();

**while** (values.iterator().hasNext()){

sb.append(","+ values.iterator().next().toString());

}

context.write(key, **new** Text (sb.toString().replaceFirst(",", "")));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

BasicConfigurator.*configure*();

Job job = Job.*getInstance*(conf); // 和hadoop交互的接口 里面配置

Path input = **new** Path(args[0]);

Path output = **new** Path(args[1]);

FileSystem fs = FileSystem.*get*(conf);

**if**(fs.exists(output))

fs.delete(output, **true**);

job.setJarByClass(A\_preProcess.**class**);

job.setMapperClass(myMapperA.**class**);

job.setReducerClass(myReducerA.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(Text.**class**);

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, output);

job.waitForCompletion(**true**);

}

}

**4.2 B\_Coocuiirence.java**

**public** **class** B\_Coocurrence { //static 代表 mapper 可以独立 不依附wordcount 别的类也可以调用

**public** **static** **class** myMapperB **extends** Mapper<LongWritable, Text, Text, IntWritable>{

@Override // input key-value pair output key-value pair

**public** **void** map(LongWritable key, Text value, Context contex) **throws** IOException, InterruptedException {

// key 不用管 value 文件的一行 中转站：HDFS 暂时存mapper的output 用context交互mapreduce和HDFS

//value.toString().split();

String line = value.toString();

String[] user\_movie\_rating = line.split("\t");

//String user = user\_movie\_rating[0];

String movieRating = user\_movie\_rating[1];

String [] movie\_rating = movieRating.split(",");

**for** (**int** i = 0; i < movie\_rating.length; i++){

String movie1 = movie\_rating[i].trim().split(":")[0];

**for** (**int** j = 0; j < movie\_rating.length; j++){

String movie2 = movie\_rating[j].trim().split(":")[0];

contex.write(**new** Text(movie1 + ":" + movie2), **new** IntWritable(1));

}

}

}

}

**public** **static** **class** myReducerB **extends** Reducer<Text, IntWritable, Text, IntWritable> {

@Override

**public** **void** reduce(Text key, Iterable<IntWritable> values,

Context context) **throws** IOException, InterruptedException {

**int** sum = 0;

**while** (values.iterator().hasNext()){

sum += values.iterator().next().get();

}

context.write(key, **new** IntWritable(sum));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration(); // 可以在这里面改配置

BasicConfigurator.*configure*();

Path input = **new** Path(args[0]);

Path output = **new** Path(args[1]);

FileSystem fs = FileSystem.*get*(conf);

**if**(fs.exists(output))

fs.delete(output, **true**);

Job job = Job.*getInstance*(conf); // 和hadoop交互的接口 里面配置

job.setJarByClass(B\_Coocurrence.**class**);

job.setMapperClass(myMapperB.**class**);

job.setReducerClass(myReducerB.**class**);

// job.setNumReduceTasks(3); // 会有三个文件为结果

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(IntWritable.**class**);

//TextInputFormat.setInputPaths(job, new Path(args[0]));

//TextOutputFormat.setOutputPath(job, new Path(args[1]));

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, output);

job.waitForCompletion(**true**);

}

}

**4.3 C\_matrixProduct.java**

**public** **class** C\_matrixProduct {

**public** **static** **class** myMapperC **extends** Mapper<LongWritable, Text, Text, DoubleWritable>{

Map<Integer, List<MovieRelation>> relations = **new** HashMap<Integer, List<MovieRelation>>();

Map<Integer, Integer> normalization = **new** HashMap<Integer,Integer>();

@Override

**public** **void** setup(Context context) **throws** IOException{

Configuration conf= context.getConfiguration();

String filePath = conf.get("coOccurrencePath"); // driver .set() B's output

Path path = **new** Path(filePath);

FileSystem fs = FileSystem.*get*(conf);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fs.open(path))); // 把文件一行一行读进来

String line = br.readLine();

**while** (line != **null**){

// movie1 : movie2 \t relation

String[] tokens = line.toString().trim().split("\t");

String [] movies = tokens[0].split(":");

**int** movie1 =Integer.*parseInt*(movies[0]);

**int** movie2 = Integer.*parseInt*(movies[1]);

**int** relation = Integer.*parseInt*(tokens[1]);

MovieRelation mr = **new** MovieRelation(movie1, movie2, relation);

**if** (relations.containsKey(movie1)){

relations.get(movie1).add(mr);

}

**else** {

List<MovieRelation> l = **new** ArrayList<MovieRelation>();

l.add(mr);

relations.put(movie1, l);

}

line = br.readLine();

}

br.close();

// set up normalization

**for** (Map.Entry<Integer, List<MovieRelation>> entry : relations.entrySet()){

**int** sum = 0;

**for** (MovieRelation mr :entry.getValue()){

sum += mr.getRelation();

}

normalization.put(entry.getKey(), sum);

}

}

@Override // input key-value pair output key-value pair

**public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {

// user- movie-rating

// user:movie score

String[] tokens = value.toString().trim().split(","); //最初的dataset

**int** user = Integer.*parseInt*(tokens[1]);

**int** movie = Integer.*parseInt*(tokens[0]);

**double** rating = Double.*parseDouble*(tokens[2]);

**for** (MovieRelation movieR : relations.get(movie)){

**double** score = rating \* movieR.getRelation();

// normalize

score = score / normalization.get(movieR.getMovieB());// normalization

DecimalFormat df = **new** DecimalFormat("#.00");

score = Double.*valueOf*(df.format(score));

// user - movie:score

// context.write(new IntWritable(user), new Text(movieR.getMovieB() + ":" + score));

// \_\_> user:movie - score

context.write(**new** Text(user + ":" + movieR.getMovieB()), **new** DoubleWritable(score));

}

}

}

**public** **static** **class** myReducerC **extends** Reducer<Text, DoubleWritable, IntWritable, Text> {

// input user：movie rating

// output user movie: rating sum

@Override

**public** **void** reduce(Text key, Iterable<DoubleWritable> values,

Context context) **throws** IOException, InterruptedException {

**double** sum = 0;

**while** (values.iterator().hasNext()){

sum += values.iterator().next().get();

}

String[] tokens = key.toString().split(":");

**int** user = Integer.*parseInt*(tokens[0]);

context.write(**new** IntWritable(user), **new** Text(tokens[1] + ":" + sum));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration(); // 可以在这里面改配置

conf.set("coOccurrencePath ", args[0]);

Job job = Job.*getInstance*(conf); // 和hadoop交互的接口 里面配置

job.setJarByClass(C\_matrixProduct.**class**);

job.setMapperClass(myMapperC.**class**);

job.setReducerClass(myReducerC.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setMapOutputKeyClass(Text.**class**);

job.setMapOutputValueClass(DoubleWritable.**class**);

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(Text.**class**);

//TextInputFormat.setInputPaths(job, new Path(args[1])); // 最初的raw

//TextOutputFormat.setOutputPath(job, new Path(args[2]));

FileInputFormat.*addInputPath*(job, **new** Path(args[1]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[2]));

job.waitForCompletion(**true**);

}

}

**4.4 D\_recommandDetail.java**

**public** **class** D\_RecomandDetail {

**public** **static** **class** myMapperD **extends** Mapper<LongWritable, Text, IntWritable, Text>{

// filter watched movies

Map<Integer, List<Integer>> watchedHistory = **new** HashMap<Integer, List<Integer>>();

@Override

**public** **void** setup(Context context) **throws** IOException{

//read movie watched history

Configuration conf= context.getConfiguration();

String filePath = conf.get("watchedHistory"); // driver .set() 最初的data

Path path = **new** Path(filePath);

FileSystem fs = FileSystem.*get*(conf);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fs.open(path))); // 把文件一行一行读进来

String line = br.readLine();

// movie user rating

**while** (line != **null**){

// movie1 : movie2 \t relation

**int** userid =Integer.*parseInt*(line.split(",")[1]);

**int** movieid= Integer.*parseInt*(line.split(",")[0]);

**if** (watchedHistory.containsKey(userid)){

watchedHistory.get(userid).add(movieid);

}

**else** {

List<Integer> l = **new** ArrayList<Integer>();

l.add(movieid);

watchedHistory.put(userid, l);

}

line = br.readLine();

}

br.close();

}

@Override

**public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {

// filtering

// user movie : rating

String[] tokens = value.toString().trim().split("\t");

**int** user = Integer.*parseInt*(tokens[0]);

**int** movie = Integer.*parseInt*((tokens[1].split(":")[0]).trim());

**if** (!watchedHistory.get(user).contains(movie)){

context.write(**new** IntWritable(user), **new** Text(tokens[1]));

}

}

}

// match movie\_name and movie id

**public** **static** **class** myReducerD **extends** Reducer<IntWritable, Text, IntWritable, Text> {

Map<Integer, String> movieTitle = **new** HashMap<Integer, String>();

@Override

**protected** **void** setup(Context context) **throws** IOException{

Configuration conf= context.getConfiguration();

String filePath = conf.get("movie\_titles"); // driver .set() title data

Path path = **new** Path(filePath);

FileSystem fs = FileSystem.*get*(conf);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fs.open(path))); // 把文件一行一行读进来

String line = br.readLine();

// movieid, year, movietitle

**while**(line != **null**){

**int** movie\_id = Integer.*parseInt*(line.trim().split(",")[0]);

movieTitle.put( movie\_id, line.trim().split(",")[2]);

line = br.readLine();

}

br.close();

}

@Override

**public** **void** reduce(IntWritable key, Iterable<Text> values,

Context context) **throws** IOException, InterruptedException {

// key -> user value -> movie : rating

**while** (values.iterator().hasNext()){

String cur = values.iterator().next().toString();

**int** movieid = Integer.*parseInt*(cur.split(":")[0]);

String rating = cur.split(":")[1];

context.write(key, **new** Text(movieTitle.get(movieid) + ":" + rating));

// userid movie name : rating

}

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration(); // 可以在这里面改配置

conf.set("watchedHistory", args[0]); // 最初的data

conf.set("movie\_titles", args[1]); // title's data

Job job = Job.*getInstance*(conf); // 和hadoop交互的接口 里面配置

job.setJarByClass(D\_RecomandDetail.**class**);

job.setMapperClass(myMapperD.**class**);

job.setReducerClass(myReducerD.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

//job.setMapOutputKeyClass(IntWritable.class);

//job.setMapOutputValueClass(Text.class);

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(Text.**class**);

//TextInputFormat.setInputPaths(job, new Path(args[2])); // C's result

//TextOutputFormat.setOutputPath(job, new Path(args[3]));

FileInputFormat.*addInputPath*(job, **new** Path(args[2]));

FileOutputFormat.*setOutputPath*(job, **new** Path(args[3]));

job.waitForCompletion(**true**);

}

}

**4.5 E\_invertedIndex**

**public** **class** E\_InvertedIndex {

**public** **static** **class** myMapper **extends** Mapper<LongWritable, Text, Text, Text>{

List<String> stop = **new** ArrayList<String>();

@Override

**public** **void** setup(Context context) **throws** IOException{

Configuration conf = context.getConfiguration();

String filePath = conf.get("StopPath");// /input/stopWord.txt

Path path = **new** Path(filePath);

//Path path = new Path("hdfs:" + filePath); // load file to local from HDFS hdfs:/input/stopwords.txt

FileSystem fs = FileSystem.*get*(conf);

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fs.open(path)));

String line;

line = br.readLine();

**while** (line !=**null**){

stop.add(line.toLowerCase().trim());

line = br.readLine();

}

}

@Override

**public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {

String[] movie\_info = value.toString().trim().split(",");

String movieId = movie\_info[0];

Text id = **new** Text(movieId);

String names = movie\_info[2];

StringTokenizer st = **new** StringTokenizer(names);

**while** (st.hasMoreTokens()){

String cur = st.nextToken().toLowerCase().trim();

cur = cur.replaceAll("[^a-zA-Z]", "");

**if** (!stop.contains(cur) && cur.trim() != **null** && cur.trim() != ""){

context.write(**new** Text(cur), id);

}

}

}

}

**public** **static** **class** myReducer **extends** Reducer<Text,Text, Text, Text> {

**public** **void** reduce(Text key, Iterable<Text> values,

Context context) **throws** IOException, InterruptedException {

// key : keyword values : {doc1, doc1, doc1, doc2, doc2}

// \_\_> key: keyword value : doc1\doc2

String last = **null**;

StringBuilder movieList = **new** StringBuilder();

//int count = 0; // keyword 在 doc出现的cont

//int threshold = 1;//100 keyword在doc里出现100次才行

**for** (Text value: values){

**if** (last != **null** && value.toString().trim().equals(last)){

//count++;

**continue**;

}

// lastbook == null

**if** (last == **null**){

// count ++;

last = value.toString().trim();

**continue**;

}

movieList.append(last); // if last != null && count > threshold

movieList.append("\t");

//count = 1;

last = value.toString().trim();

}

//if (threshold < count){

movieList.append(last);// 最后一个

//}

**if** (!movieList.toString().trim().equals("") && ! key.toString().equals("")){

context.write(key, **new** Text (movieList.toString())); // 从mapreduce中写到hdfs

}

}// reduce

} // Reducer

**public** **static** **void** main(String[] args) **throws** Exception {

**if** (args.length < 3){

**throw** **new** Exception("Usage: <input dir> <output dir> <stopwords dir>");

}

Configuration conf = **new** Configuration(); // 可以在这里面改配置

BasicConfigurator.*configure*();

conf.set("StopPath", args[0]); // 读进来

Job job = Job.*getInstance*(conf); // 和hadoop交互的接口 里面配置

Path input = **new** Path(args[1]);

Path output = **new** Path(args[2]);

FileSystem fs = FileSystem.*get*(conf);

**if**(fs.exists(output))

fs.delete(output, **true**);

job.setJarByClass(E\_InvertedIndex.**class**);

job.setMapperClass(myMapper.**class**);

job.setReducerClass(myReducer.**class**);

//job.setNumReduceTasks(3); // will generate 3 output files

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(Text.**class**);

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, output); // path 是一个文件夹的path

System.*exit*(job.waitForCompletion(**true**)? 0 : 1);

}

}

**4.6 F\_FileJoin.java**

**public** **class** F\_FileJoin{

**public** **static** **class** myMapperF **extends** Mapper<LongWritable, Text, IntWritable, Text> {

**private** Map<Integer, String> movieName = **new** HashMap<Integer, String>();

@Override

**protected** **void** setup(Context context) **throws** IOException{

Configuration conf= context.getConfiguration();

String filePath = conf.get("movie\_titles"); // driver .set() title data

Path path = **new** Path(filePath);

FileSystem fs = FileSystem.*get*(conf); // hdfs:/

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(fs.open(path))); // 把文件一行一行读进来

// FSDataInputStream fsInput = fs.open(new Path(filePath)); //

//String line = fsInput.readLine(); //

String line = br.readLine();

// movieid, year, movietitle

**while**(line != **null**){

**int** movie\_id = Integer.*parseInt*(line.trim().split(",")[0]);

movieName.put(movie\_id, line.trim().split(",")[2]);

//line = br.readLine();

line = br.readLine();

}

//br.close();

br.close();

}

**public** **void** map(LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String[] tokens = value.toString().split(","); // 最初的data

//int usrid = Integer.parseInt(tokens[1].trim());

**int** movieid = Integer.*parseInt*(tokens[0].trim());

String rating = tokens[2].trim();

**if** (movieName.containsKey(movieid)) {

//oKey.set(tokens[0].trim()); // movieid

//oValue.set(movieName.get(movieid) + ":" + rating); // moviename :rating

context.write(**new** IntWritable(movieid), **new** Text(movieName.get(movieid) + "@" + rating));

}

}

}

**public** **static** **class** myReducerF **extends** Reducer<IntWritable, Text, IntWritable, Text> {

@Override

**public** **void** reduce(IntWritable key, Iterable<Text> values, Context context)

**throws** IOException, InterruptedException {

**long** sum = 0;

**int** count = 0;

String moviename ="";

**for** (Text value : values) {

String cur = value.toString();

moviename = cur.split("@")[0].trim();

**double** rate = Double.*parseDouble*(cur.split("@")[1].trim());

sum += rate;

count++;

}

**double** average = (**double**) sum / count;

DecimalFormat df = **new** DecimalFormat("#.00");

average = Double.*valueOf*(df.format(average));

context.write(key, **new** Text("@" + moviename + "@" + average));

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

BasicConfigurator.*configure*();

conf.set("movie\_titles", args[0]); //

Job job = Job.*getInstance*(conf); //

Path input = **new** Path(args[1]);

Path output = **new** Path(args[2]);

FileSystem fs = FileSystem.*get*(conf);

**if**(fs.exists(output))

fs.delete(output, **true**);

job.setJarByClass(F\_FileJoin.**class**);

job.setMapperClass(myMapperF.**class**);

job.setReducerClass(myReducerF.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setMapOutputKeyClass(IntWritable.**class**);

job.setMapOutputValueClass(Text.**class**);

job.setOutputKeyClass(IntWritable.**class**);

job.setOutputValueClass(Text.**class**);

//TextInputFormat.setInputPaths(job, new Path(args[1])); // 最初的data

//TextOutputFormat.setOutputPath(job, new Path(args[2]));

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, output);

job.waitForCompletion(**true**);

}

}

**4.7 G\_TopK.java**

**public** **class** G\_TopK {

**static** **class** MyMap **extends** Mapper<LongWritable, Text, NullWritable, Text> {

**public** **static** **final** **int** ***K*** = 100;

**private** TreeMap<Double, Text> averages = **new** TreeMap<Double, Text>();

**public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {

// movid moviName : average

String line = value.toString();

String[] tokens = line.split("@");

**double** average = Double.*parseDouble*(tokens[2].trim());

averages.put(average, **new** Text(value));

**if** (averages.size() > ***K***)

averages.remove(averages.firstKey());

}

@Override

**protected** **void** cleanup(Context context) **throws** IOException, InterruptedException {

**for** (Text value : averages.values()) {

context.write(NullWritable.*get*(), value);

}

}

}

**static** **class** MyReduce **extends** Reducer<NullWritable, Text, NullWritable,Text> {

**public** **static** **final** **int** ***K*** = 100;

**private** TreeMap<Double, Text> averages = **new** TreeMap<Double, Text>();

@Override

**protected** **void** reduce(NullWritable key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {

**for** (Text value : values) {

String[] tokens = value.toString().split("@");

**double** average = Double.*parseDouble*(tokens[2].trim());

averages.put(average, **new** Text(value));

**if** (averages.size() > ***K***){

averages.remove(averages.firstKey());

}

}

**int** i = 0;

**for** (Text t : averages.descendingMap().values()) {

i++;

// Output our ten records to the file system with a null key

String line = t.toString();

String [] tokens = line.split("@");

context.write(NullWritable.*get*(), **new** Text(i +". " + tokens[0] + " " + tokens[1] + " @ " + tokens[2]));

}

}

} // reducer

**public** **static** **void** main(String[] args) **throws** Exception {

Configuration conf = **new** Configuration();

BasicConfigurator.*configure*();

Job job = **new** ~~Job~~(conf, "top100");

Path input = **new** Path(args[0]);

Path output = **new** Path(args[1]);

FileSystem fs = FileSystem.*get*(conf);

**if**(fs.exists(output))

fs.delete(output, **true**);

job.setJarByClass(G\_TopK.**class**);

job.setMapperClass(MyMap.**class**);

job.setReducerClass(MyReduce.**class**);

//job.setNumReduceTasks(3);

job.setOutputKeyClass(NullWritable.**class**);

job.setOutputValueClass(Text.**class**);

job.setInputFormatClass(TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

FileInputFormat.*addInputPath*(job, input);

FileOutputFormat.*setOutputPath*(job, output); // path 是一个文件夹的path

job.waitForCompletion(**true**);

}

}

**4.8 MovieRelation.java**

**public** **class** MovieRelation {

**public** **int** movieA;

**public** **int** movieB;

**public** **int** relation;

**public** MovieRelation(**int** a, **int** b, **int** r){

movieA= a;

movieB = b;

relation = r;

}

**public** **int** getMovieA() {

**return** movieA;

}

**public** **void** setMovieA(**int** movieA) {

**this**.movieA = movieA;

}

**public** **int** getMovieB() {

**return** movieB;

}

**public** **void** setMovieB(**int** movieB) {

**this**.movieB = movieB;

}

**public** **int** getRelation() {

**return** relation;

}

**public** **void** setRelation(**int** relation) {

**this**.relation = relation;

}

}

1. **Reference**

MapReduce Design Patterns, By Donald Miner, Adam Shook, November 2012

<http://academictorrents.com/details/9b13183dc4d60676b773c9e2cd6de5e5542cee9a>

<http://courses.cs.washington.edu/courses/csep546/10sp/hw1/>

<http://www.bkjia.com/yjs/951260.html>