DS503 Project 1 Report

Wenhan Ji, Chen Ding

Contents

1. Creating datasets

Following the descriptions of Facebook-like applications, we created the datasets with detailed and meaningful strings using Python. The datasets are as below:

MyPage

	ID	Name	Nationality	CountryCode	Hobby
0	1	Brad Yarberry	Angolan	6	I dislike Taxidermy
1	2	Ted Conover	Bahamian	13	I dislike Basketball

Friends

	FriendRel	PersonID	MyFriend	DateofFriendship	Desc
0	1	36156	82603	147927	collegefriend
1	2	20301	71718	666716	girlfriend

AccessLog

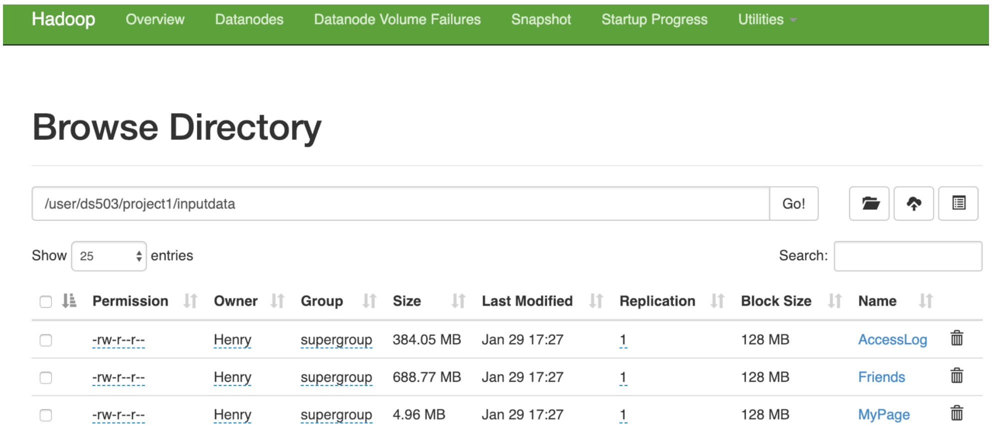
	AccessID	ByWho	WhatPage	TypeOfAccess	AccessTime
0	1	96024	88567	watch live	769323
1	2	7570	32466	left a note	567265

2. Loading Datasets into Hadoop

We use the ***-put*** command line as below in the Terminal to load datasets into HDFS.

hdfs dfs -put inputdata /user/ds503/project1/inputdata

Show the dataset files in the HDFS Web User Interface as below:



3. Accomplishing Analytics Tasks using MapReduce Jobs

a. Task a

We write a map-only Java program to do the data filtering for the MyPage dataset. The Mapper function reads the MyPage text file line by line, and find the nationality information of that line. If the nationality is the same as our own Nationality, Chinese, the Mapper function will write the line to the value of the output, and we also set the key of the output as the NullWritable.get(). A Combiner is unnecessary in this MapReduce job because it's a map-only job in which we just filter out the target line, and we don't need to aggregate what are filtered.

We use the ***hadoop jar*** command below to run the job using TaskA.jar. The inputdata is MyPage and output the result into directory TaskA.

hadoop jar TaskA.jar /user/ds503/project1/inputdata/MyPage /user/ds503/project1/TaskA

The output of our program is shown as below:

```
56,William Bee,Chinese,40,I like Glassblowing
81,Sheldon Payne,Chinese,40,I like Lockpicking
82,Toby Griffin,Chinese,40,I dislike tabletop games
254,Sylvia Tobin,Chinese,40,I dislike Sports
293,Toni Adams,Chinese,40,I like Baseball
341,James Lyles,Chinese,40,I dislike Cooking
362,Jerry Greenwood,Chinese,40,I dislike Amateur radio
453,David Dingle,Chinese,40,I dislike amateur radio
643,Donald Johnson,Chinese,40,I like Brazilian jiu-jitsu
676,Barbara Barrientes,Chinese,40,I like Homebrewing
755,Charles Post,Chinese,40,I like Parkour
852,Deanna Villalpando,Chinese,40,I dislike Urban exploration
994,William Ladue,Chinese,40,I dislike Fashion
1006,Karen Carlson,Chinese,40,I dislike Do it yourself
```

b. Task b

This task is quite similar to the WordCount task. We write a Map function and to find the country information in each line and write a Reduce function to sum up the number of pair from each country. The details of functions are as below:

Step 1: Map function

Map Input key: file line number

Map Input value: file line text

Map Output key: countryCode

Map Output value: 1

Step 2: Shuffling and Sorting phase

Sort and merge the output pairs from Map function by key.

Step 3: Reduce function

Reduce Input: countryCode, [1, 1, 1, 1]

Reduce Output Key: countryCode

Reduce Output Value: sum over the input value of list and get the citizen count of each country.

In this WordCount-like MapReduce task, we use the Reducer as the Combiner for two reasons. First, the Reducer function performs the aggregation functionality. Then, the Combiner and Reducer function have the same input and output.

We use the ***hadoop jar*** command below to run the job using TaskB.jar. The inputdata is MyPage and output the result into directory TaskB.

hadoop jar TaskB.jar /user/ds503/project1/inputdata/MyPage /user/ds503/project1/TaskB

The output of our program is shown as below:

```
Afghan  1474
Albanian  1417
Algerian  1453
American  1400
Andorran  1388
Angolan 1420
Antiguans 1416
Argentinean 1375
Armenian  1378
Australian  1451
```

The costs of time with Combiner and without Combiner are shown below:

With Combiner	**Without Combiner**
31s	42s

c. Task c

We have two MapReduce jobs to solve the task C. The first job is just like the WordCount task, in which we filter out the access frequency of each page for the AccessLog dataset's WhatPage column. The second job read the output of the first job as the input and summarize the Top 10 interesting page.

The first job:

Output: <key, value> = <pageID, accessedCount>

The second job:

Map input: <key, value> = <pageID, accessedCount>

Map output: <key, value> = <accessedCount, page ID>

Sort merge phase: sort the key reversely.

Reduce: write out the first 10 <key, value> pairs read in, which means selecting out the first 10 interesting page.

The first MapReduce job is like WordCount, so the Reducer is also used as the Combiner. In the second MapReduce job, a Combiner is not necessary. The key point in our design is the Comparator.class, in which we defines the shuffling-sorting phase into a descending order, so that we can find the top 10 in the reducer. In this case, our Mapper in this task is just change the place of key and value in the <key, value> pair to make it convenient to be descending sorted. Thus, the output from Mapper function does not need to be aggregated nor need to be selected pairs with top value.

We use the ***hadoop jar*** command below to run the job using TaskC.jar. The inputdata is AccessLog and output the result into directory TaskC.

hadoop jar TaskC.jar /user/ds503/project1/inputdata/AccessLog /user/ds503/project1/TaskC

The output of our program is shown as below:

Output of the First job: <PageID, AccessCount>

```
1 97
10  112
100 81
1000  104
10000 93
100000  97
10001 103
10002 107
10003 108
10004 110
```

Output of the Second job: <PageID, AccessCount>

```
71676 153
23829 147
52766 145
14749 143
44898 143
35515 143
74077 142
7884  142
92766 141
64136 140
```

The costs of time with Combiner and without Combiner are shown below:

	With Combiner	**Without Combiner**
Job 1	63s	71s
Job 2	-	-

d. Task d

We design two MapReduce jobs and use two datasets Friends and MyPage.

The first job is to count the friend number of each person. It's similar with wordcount but we need to make a slightly change. The ID in personID and MyFriend all need to be counted. Below is how we do it.

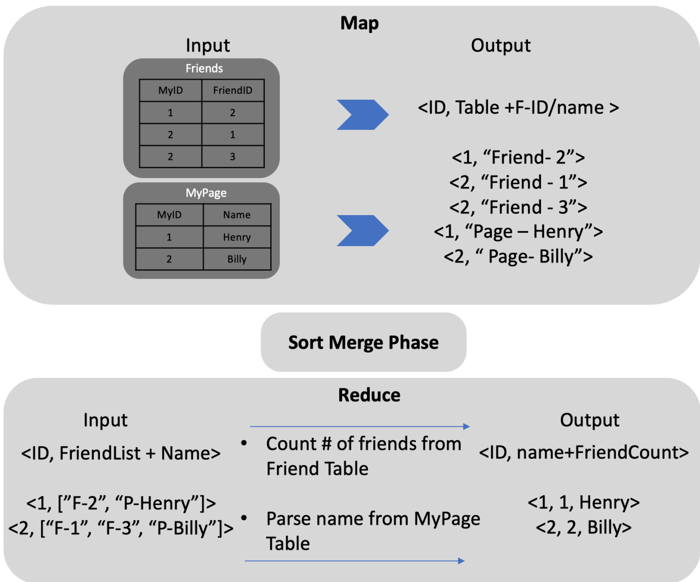
For each Friend Relation, in mapper output step, we write the key-id both of the person who add others and the person who is added.

```
Text id1 = new Text(lineList[1]);
Text id2 = new Text(lineList[2]);
context.write(id1, new IntWritable( value: 1));
context.write(id2, new IntWritable( value: 1));
```

Outout: <id, friendCount>

The second job is to join the Job1's output dataset<id, friendCount> with MyPage dataset which<id, name> information. Their common key is ID and ID is unique in both dataset.

Below is how we apply reduce side join.



We design a combiner to count the number of friends from the outputs of Mapper in each node. To realize the goal, we have to skip the output pairs from MyPage datasets and only make the combiner performs on the output pairs from Friend datasets.

We use the ***hadoop jar*** command below to run the job using TaskD.jar. The inputdata is Friends and MyPage and output the result into directory TaskD.

hadoop jar TaskD.jar /user/ds503/project1/inputdata/Friends /user/ds503/project1/inputdata/MyPage /user/ds503/project1/TaskD

The output of our program is shown as below:

Output of the First job: <ID, Count>

```
1,404
10,411
100,410
1000,423
10000,357
100000,371
10001,413
10002,395
10003,414
10004,404
```

Output of the Second job: <ID, Number of Friends, Name>

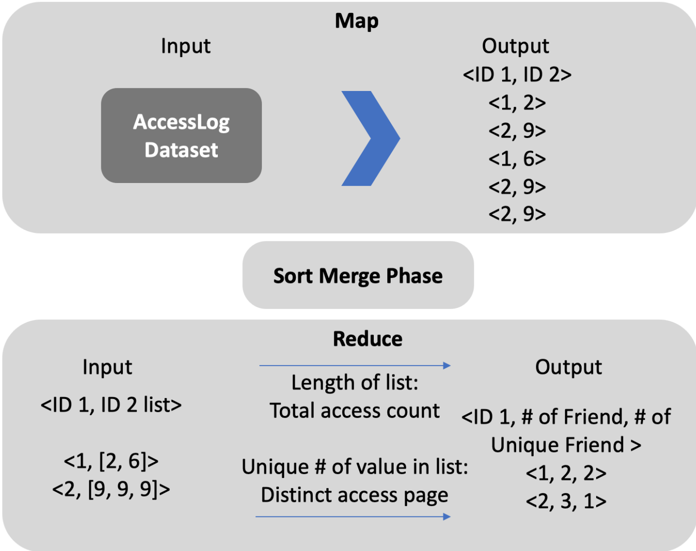
```
1,404,Brad Yarberry
2,396,Ted Conover
3,373,Timothy Barnes
4,368,Virginia Hooks
5,419,Nicole Duncan
6,364,Karen Carlson
7,410,Michele Burnside
8,416,Amy Wilbur
9,457,James Chung
10,411,Maria Bartz
11,402,Tina Kreisler
12,378,Margarete Deschamps
```

The costs of time with Combiner and without Combiner are shown below:

With Combiner	**Without Combiner**
78s	84s

e. Task e

The MapReduce job in Task E consists of a map function that selects out the PersonID and MyFriend columns as the ID1 and ID2, and a reduce function that computes the number of friend and number of unique friend. The concept framework is as below:



In Task E, even though we have aggregations from Mapper's outputs to Reducer's inputs, it is hard to design a Combiner to aggregate the outputs in each node and to make sure that the output of the Combiner can also be aggregated during the shuffling and sorting phase.

We use the ***hadoop jar*** command below to run the job using TaskE.jar. The inputdata is AccessLog and output the result into directory TaskE.

hadoop jar TaskE.jar /user/ds503/project1/inputdata/AccessLog /user/ds503/project1/TaskE

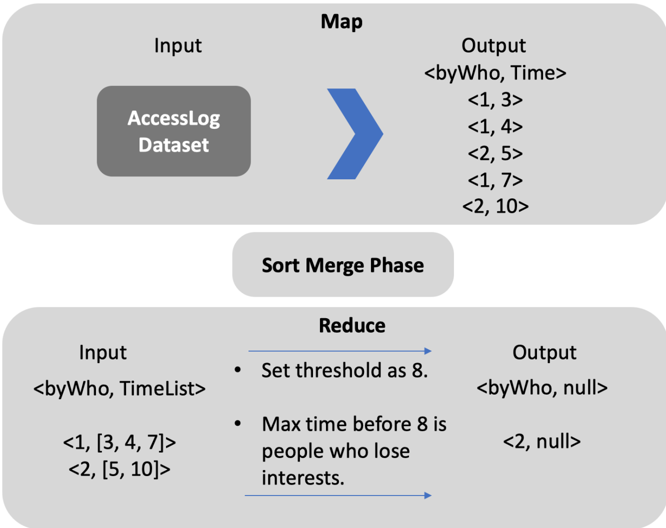
The output of our program is shown as below:

<ID 1, # of Friend, # of Unique Friend >

```
176 98  98
177 99  99
178 101 101
179 94  94
180 105 105
181 89  88
182 106 106
183 105 105
```

f. Task f

A MapReduce job has been designed to solve the task F. Mapper step selects out the <ByWho, accessTime> as key-value pairs. Reduce input key is ByWho user ID, value is a list of time the user has accessed other user's page. In reduce function, the max value of the list is the last time the user accessed. If that maxTime < thresholdTime, then he is the person who loses interest. So we write out his id as the output.



We design a Combiner to compute the maxTime in each node and use the maxTime as the output value of Combiner. Then the reducer can save time by comparing the maxTime from each node, rather than comparing all the Time records from Mapper's outputs.

We use the ***hadoop jar*** command below to run the job using TaskF.jar. The inputdata is AccessLog dataset and output the result into directory TaskF.

hadoop jar TaskF.jar /user/ds503/project1/inputdata/AccessLog /user/ds503/project1/TaskF

The output of our program is shown as below:

<byWho, null>

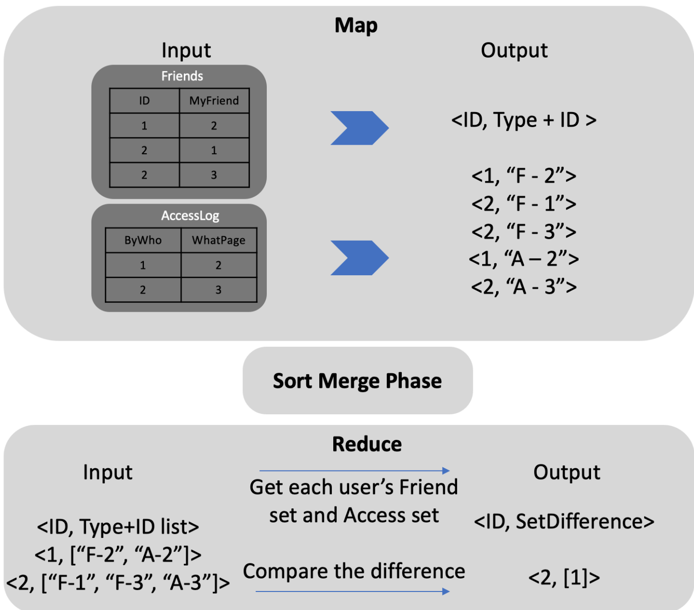
```
1   14342
2   20076
3   32608
4   87364
5   90875
```

The costs of time with Combiner and without Combiner are shown below:

With Combiner	**Without Combiner**
43s	37s

g. Task g

A MapReduce job has been designed to solve the task G using Friends and AccessLog two data sets. The mapper step selects out the <ID, Type+ID> as key-value pairs. Reduce input key is user ID and value is a list of the user's friends set and access record sets. Then we can parse the list into a Friend Set and a Access Set and compare the difference between the two sets for each user. We write the difference as the output. Here is the concept frame and example as below.



We cannot design a combiner to do the aggregation because in this task we have to get the global friendID and accessID then compute the difference. An aggregation combiner will eliminate some records.

We use the ***hadoop jar*** command below to run the job using TaskG.jar. The inputdata is Friends dataset and AccessLog dataset and output the result into directory TaskG.

hadoop jar TaskG.jar /user/ds503/project1/inputdata/Friends user/ds503/project1/inputdata/AccessLog /user/ds503/project1/TaskG

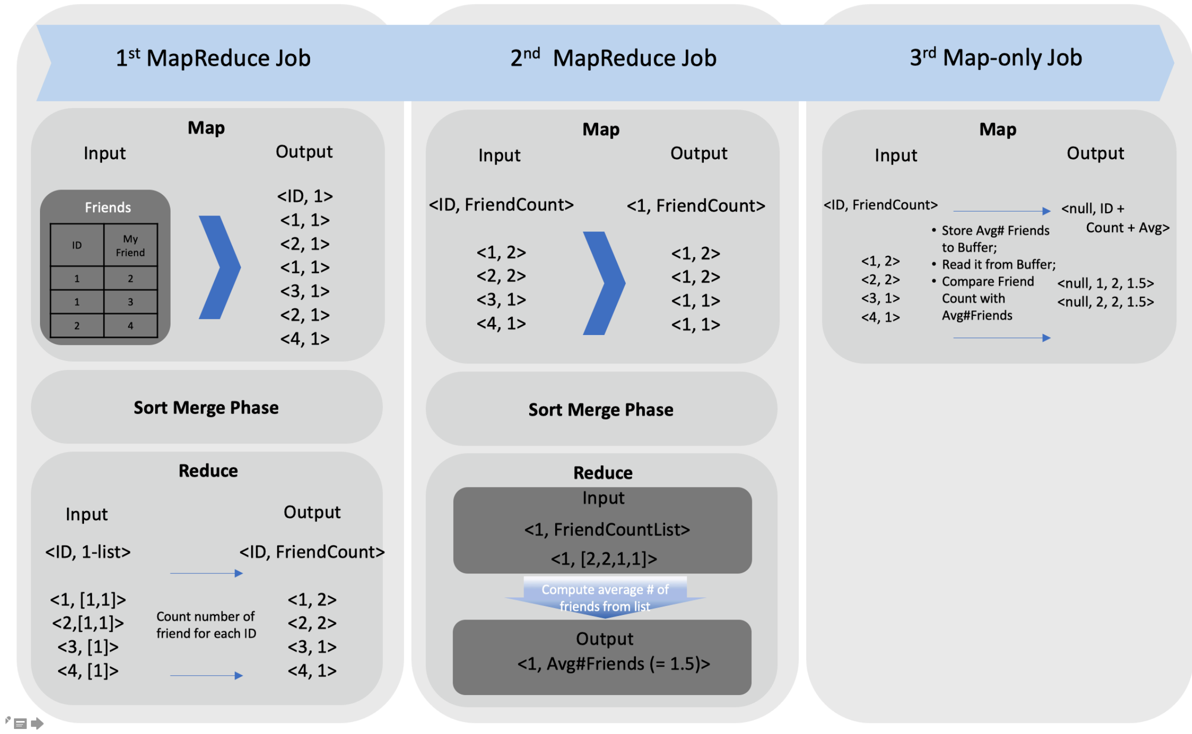
The output of our program is shown as below:

<ID: friendCount, loseInterestFriendCount>

```
20: 210,209
21: 217,217
22: 206,206
23: 202,202
24: 190,189
25: 225,225
26: 196,196
27: 210,208
28: 223,223
```

h. Task h

Three MapReduce jobs have been designed to solve Task H. Firstly, we design a WordCount-like mapreduce job to count friends number of each ID. Then, in the second MapReduce job, we compute the average number of friends among all the IDs. Lastly, we design a Map-only job and a function. The function is designed to read the average number of friends from the output of the second mapreduce job and store it into buffer. The map-only job is used to filter out the users that have friends above the average.



We use the ***hadoop jar*** command below to run the job using TaskH.jar. The inputdata is Friends dataset and we output the result into directory TaskH.

hadoop jar TaskH3.jar /user/ds503/project1/inputdata/Friends /user/ds503/project1/TaskH3

The output of our program is shown as below:

<ID, FriendCount, TotalAvg>

```
99999 418,400.0
99994 405,400.0
99993 447,400.0
99991 418,400.0
99990 414,400.0
99988 409,400.0
99986 404,400.0
99985 420,400.0
99983 422,400.0
99978 406,400.0
99975 412,400.0
```

In this task, we cannot design a combiner to leverage the efficiency of our MapReduce jobs, because we mostly need all the records from the mappers without aggregation.

4. Accomplishing Analytics Tasks Using Apache Pig

a. Task a

The query in Task A uses Filter operator to filter out the data that nationality is Chinese.

```
56  William Bee Chinese 40  I like Glassblowing
81  Sheldon Payne Chinese 40  I like Lockpicking
82  Toby Griffin  Chinese 40  I dislike tabletop games
254 Sylvia Tobin  Chinese 40  I dislike Sports
293 Toni Adams  Chinese 40  I like Baseball
341 James Lyles Chinese 40  I dislike Cooking
362 Jerry Greenwood Chinese 40  I dislike Amateur radio
453 David Dingle  Chinese 40  I dislike amateur radio
643 Donald Johnson  Chinese 40  I like Brazilian jiu-jitsu
676 Barbara Barrientes  Chinese 40  I like Homebrewing
```

b. Task b

The query first groups the MyPage dataset using the attribute Nationality, then counts the number of tuples in each group, finally we ordered the table by the attribute nationality.

<nation, count>

```
Afghan  1474
Albanian  1417
Algerian  1453
American  1400
Andorran  1388
Angolan 1420
Antiguans 1416
Argentinean 1375
Armenian  1378
Australian  1451
```

c. Task c

The query first groups the AccessLog dataset using the attribute WhatPage, then counts the number of tuples in each group. Then we ordered the table by the number of tuples in each group. Finally we use the Limit operator to get the first 10 tuples in the sorted table.

<top 10 pageID, count>

```
71676 153
23829 147
52766 145
14749 143
35515 143
44898 143
7884  142
74077 142
92766 141
45773 140
```

d. Task d

The query first generate both PersonID and MyFriend from the Friends dataset as ID to get all the friendship in the table. Then the query groups by ID and count the number of friends for each ID. Next we sort the table with two attributes the number of friend and ID. Finally we join the sorted table with MyPage on ID and get the final results.

<name, id, friendCount>

```
Brad Yarberry 1 404
Ted Conover 2 396
Timothy Barnes  3 373
Virginia Hooks  4 368
Nicole Duncan 5 419
Karen Carlson 6 364
Michele Burnside  7 410
Amy Wilbur  8 416
James Chung 9 457
Maria Bartz 10  411
```

e. Task e

First, we project ByWho and WhatPage (id1 access id2) to get id1AccessID2 dataset. Second, we distinct it to get a distinctId1AccessId2 dataset. Third, for both of two datasets, we group by id and compute the accessCount. So we can get two count, one is distinct, another is not distinct. Forth, we join by id to get these two count into one table.

<ID, totalAccessCount, totalDistinctAccessCount>

```
176 98  98
177 99  99
178 101 101
179 94  94
180 105 105
181 89  88
182 106 106
183 105 105
184 104 104
185 119 119
186 103 103
```

f. Task f

First, we group by id and get the last time each person access. Second, filter out those lastAccessTime < thresholdTime

<loseInterestID, lastAccessTime (our threshold is 900000) >

```
14342 876717
20076 894030
32608 883993
87364 890861
90875 891865
```

g. Task g

First we use groupby id to for AccessLog and Friends to compute, for each person their friends id bag and access page id bag. Second, we join these two bags according to id. Third, for each person, we do the subtract of two bags. Then we get a bag that only in friendsID bag but not accessID bag. The length of the bag is number of people he lose interest. Forth, we filter out the null bag and output.

<ID, friendCount, loseInterestCount >

```
20: 210,209
21: 217,217
22: 206,206
23: 202,202
24: 190,189
25: 225,225
26: 196,196
27: 210,208
28: 223,223
```

The person with id 24 has 190 friends totally, but 189 of them have never been accessed.

h. Task h

First, we group by friend id to count friend number for each person. Second, we use group all to turn all friend count into a bag. Third, we use aggregation function AVG to compute the mean of count and get a new table with only one average value. Finally, we filter the is with friend count > total average.

<id, friendCount, totalAvg>

```
1 404 400.0
5 419 400.0
7 410 400.0
8 416 400.0
9 457 400.0
10  411 400.0
11  402 400.0
13  413 400.0
14  414 400.0
15  427 400.0
17  402 400.0
```

5. Contribution

Wenhan Ji	**Chen Ding**
• Discuss and create datasets • Write Java codes to accomplish the analytics tasks • Run the codes and screenshot the outputs • Monitor the performance with combiner or not	• Discuss and create datasets • Analyze tasks; figure out what to do • Load datasets into Hadoop • Write Pig to accomplish the analytics tasks • Draw concept graphs and write reporßts