PROJECT 1:
Who is Most Popular?

DS503
Big Data Management
Team 1

Author:
Han Jiang
Xiaoting Cui
Dennis Silva

Supervisor:
Dr.Rundensteiner

September 22, 2017

Preface:

The following code was written and ran within the already-built vm provided by meltabakh.

Note: Make sure you start hadoop first!

CREATE A HDFS DIRECTORY
if you want to make a new directory to put code in use:
>> hadoop dfs -mkdir <new_path>

for example:
>> hadoop dfs -mkdir /user/hadoop/pigdata

DOWNLOAD OUTPUT
once you complete a job and store your output, to download output file stored
in hdfs to local system use:
>> hadoop dfs -get /user/hadoop/<output_file_name> <local_path>

for example:
>> hadoop dfs -get /user/hadoop/sampleOutput /home/hadoop/Documents/Project1-Pig

ADD A FILE/FOLDER
to add a file to the hdfs (file to be read later)
>> sudo hadoop dfs -put <input_file_path> <hdfs_output_file_path>

for example:
>> sudo hadoop dfs -put /home/hadoop/Downloads/sampleFriends.csv /pigdata

DELETE A FILE/FOLDER
to delete/remove a file/folder from hdfs:

>> hadoop fs -rmr <remove_directory_folder>

for example:
>> hadoop fs -rmr /user/hadoop/sampleOutput

Hadoop variable:
When we want to write Int (float, text) into hadoop, we use IntWritable (FloatWritable, TextWritable).

Combiner:
The general idea of combiner is to have local reducer which can reduce the shuffle task for hadoop, resulting high efficiency. However, when some node is busy, some mapper will skip the combiner process, directly go to reducer. In this way, combiner should have the same input format and output format, both same as mapper output. When just summing up, it is good to have combiner, but in other cases, combiner may not be suitable.

1. Creating Datasets

Java Program: AccessLog.java, AccessLogWirter.java, Friends.java, FriendsWriter.java, MyPage.java, MyPageWriter.java, CreateData.java

AccessLog.java, Friends.java, MyPage.java are used to generate 3 different classes required in this task.; AccessLogWirter.java, FriendsWriter.java, and MyPageWriter.java are used to define 3 tables, and CreateData.java is to create 3 datasets.

Go to the directory containing the code:
>mkdir task1
>javac -classpath /usr/share/hadoop/hadoop-core-1.2.1.jar -d task1 ./AccessLog.java ./AccessLogWriter.java ./Friends.java ./FriendsWriter.java ./MyPage.java ./MyPageWriter.java ./createData.java

Go to task1 which containing class
>cd task1
Execute
>java createData

Then, we can get our three datasets: MyPage, Friends, and AccessLog

2. Loading Datasets to Hadoop

Create new directory:
> hadoop fs -mkdir /user/hadoop/Proj1/data
Upload data:
> cd
> hadoop fs -put proj1/javacode/task1/Friends.csv /user/hadoop/Proj1/data
> hadoop fs -put proj1/javacode/task1/MyPage.csv /user/hadoop/Proj1/data
> hadoop fs -put proj1/javacode/task1/AccessLog.csv /user/hadoop/Proj1/data

Contents of directory /user/hadoop/Proj1/data

Goto : /user/hadoop/Proj1/data go

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
AccessLog.csv	file	576 MB	3	64 MB	2017-09-21 18:26	rw-r--r--	hadoop	hadoop
Friends.csv	file	1.22 GB	3	64 MB	2017-09-21 18:25	rw-r--r--	hadoop	hadoop
MyPage.csv	file	5.34 MB	3	64 MB	2017-09-21 18:26	rw-r--r--	hadoop	hadoop

Go back to DFS home

3. Accomplishing Analytics Tasks using MapReduce Jobs

task3a.)
We only use one mapper to read MyPage dataset line by line, generating key(ID)-value(Nationality). And see if Nationality is equal to what we have chosen.
We did not use combiner because we do no need to put things together.

-- sample execution:
> hadoop jar ./project1.jar my.twoa /user/hadoop/Proj1/data/MyPage.csv /user/hadoop/Proj1/javacode_a

Result:

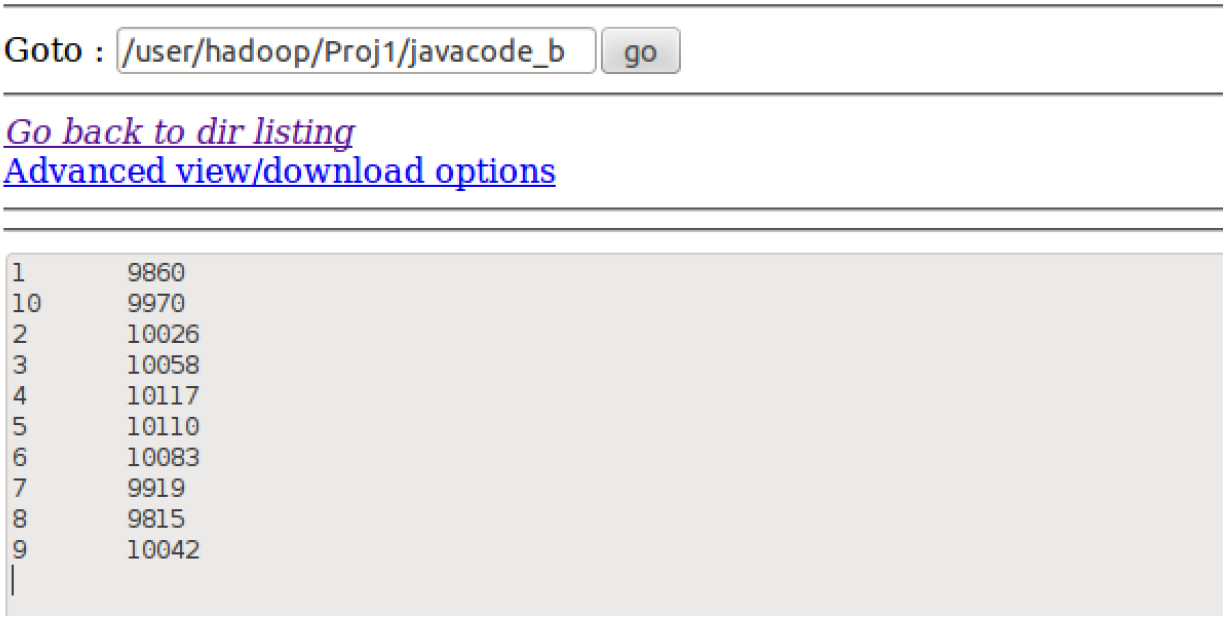
For this task, it used 210 ms.

```
17/09/22 11:31:36 INFO mapred.JobClient:   FileSystemCounters
17/09/22 11:31:36 INFO mapred.JobClient:     HDFS_BYTES_READ=5599726
17/09/22 11:31:36 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=55295
17/09/22 11:31:36 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=30
17/09/22 11:31:36 INFO mapred.JobClient:   File Input Format Counters
17/09/22 11:31:36 INFO mapred.JobClient:     Bytes Read=5599606
17/09/22 11:31:36 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 11:31:36 INFO mapred.JobClient:     Map input records=100000
17/09/22 11:31:36 INFO mapred.JobClient:     Physical memory (bytes) snapshot=43
245568
17/09/22 11:31:36 INFO mapred.JobClient:     Spilled Records=0
17/09/22 11:31:36 INFO mapred.JobClient:     CPU time spent (ms)=210
17/09/22 11:31:36 INFO mapred.JobClient:     Total committed heap usage (bytes)=
16252928
17/09/22 11:31:36 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=347
430912
17/09/22 11:31:36 INFO mapred.JobClient:     Map output records=1
```

Task3b.)
We use one mapper to read MyPage dataset line by line, generating key(CountryCode)-value(1) pairs. And one reducer takes the output key-value pairs from mapper to group by the CountryCode, counting how many persons are there.
We can use combiner, because we only need reducer to sum up, and when using combiner, it is faster.

With combiner:
-- sample execution:
> hadoop jar ./project1.jar my.twob /user/hadoop/Proj1/data/MyPage.csv /user/hadoop/Proj1/javacode_b

Result:

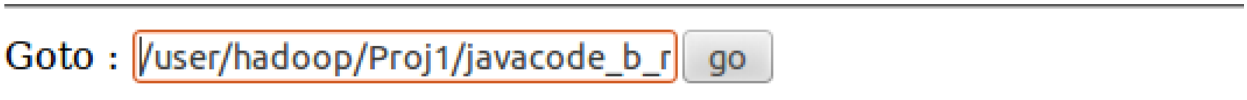


In this task, it used 610ms.

```
17/09/22 11:34:14 INFO mapred.JobClient:   File Input Format Counters
17/09/22 11:34:14 INFO mapred.JobClient:     Bytes Read=5599606
17/09/22 11:34:14 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 11:34:14 INFO mapred.JobClient:     Map output materialized bytes=87
17/09/22 11:34:14 INFO mapred.JobClient:     Map input records=100000
17/09/22 11:34:14 INFO mapred.JobClient:     Reduce shuffle bytes=87
17/09/22 11:34:14 INFO mapred.JobClient:     Spilled Records=20
17/09/22 11:34:14 INFO mapred.JobClient:     Map output bytes=609970
17/09/22 11:34:14 INFO mapred.JobClient:     Total committed heap usage (bytes)=
132190208
17/09/22 11:34:14 INFO mapred.JobClient:     CPU time spent (ms)=610
17/09/22 11:34:14 INFO mapred.JobClient:     Combine input records=100000
17/09/22 11:34:14 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
17/09/22 11:34:14 INFO mapred.JobClient:     Reduce input records=10
17/09/22 11:34:14 INFO mapred.JobClient:     Reduce input groups=10
17/09/22 11:34:14 INFO mapred.JobClient:     Combine output records=10
17/09/22 11:34:14 INFO mapred.JobClient:     Physical memory (bytes) snapshot=18
7875328
17/09/22 11:34:14 INFO mapred.JobClient:     Reduce output records=10
```

Without combiner:
-- sample execution:
> hadoop jar ./project1.jar my.twob_nocombiner /user/hadoop/Proj1/data/MyPage.csv /user/hadoop/Proj1/javacode_b_nc

Result:




```
1       9860
10      9970
2       10026
3       10058
4       10117
5       10110
6       10083
7       9919
8       9815
9       10042
```

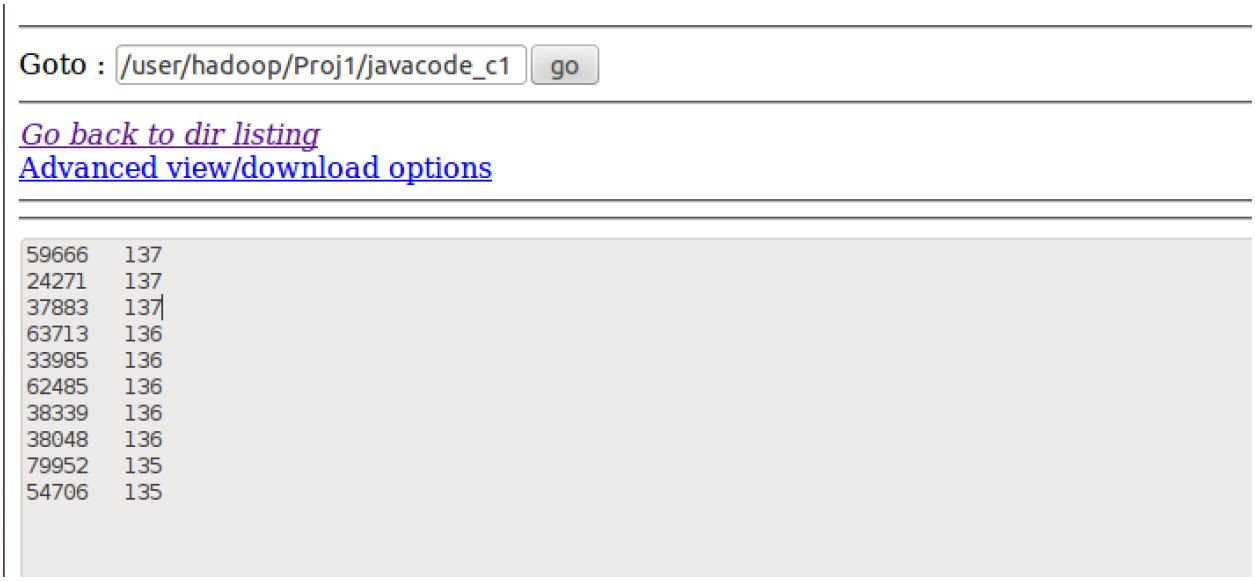
For this task, it used 690ms.

```
17/09/22 11:38:41 INFO mapred.JobClient:     Map input records=100000
17/09/22 11:38:41 INFO mapred.JobClient:     Reduce shuffle bytes=809976
17/09/22 11:38:41 INFO mapred.JobClient:     Spilled Records=200000
17/09/22 11:38:41 INFO mapred.JobClient:     Map output bytes=609970
17/09/22 11:38:41 INFO mapred.JobClient:     Total committed heap usage (bytes)=
132190208
17/09/22 11:38:41 INFO mapred.JobClient:     CPU time spent (ms)=690
17/09/22 11:38:41 INFO mapred.JobClient:     Combine input records=0
17/09/22 11:38:41 INFO mapred.JobClient:     SPLIT_RAW_BYTES=120
17/09/22 11:38:41 INFO mapred.JobClient:     Reduce input records=100000
17/09/22 11:38:41 INFO mapred.JobClient:     Reduce input groups=10
17/09/22 11:38:41 INFO mapred.JobClient:     Combine output records=0
17/09/22 11:38:41 INFO mapred.JobClient:     Physical memory (bytes) snapshot=19
2991232
17/09/22 11:38:41 INFO mapred.JobClient:     Reduce output records=10
```

Task3c.)
We use one mapper to read AccessLog dataset line by line, generation key(WhatPage)-value(1) pairs. And one reducer takes the output key-value pairs from mapper to group by WhatPage, using PriorityQuene and Comparator to sort and return top 10 values.
We cannot use combiner, because some of the mapper may not use combiner before reducer, resulting the hashtable being strange.

-- sample execution:
> hadoop jar ./task3c.jar task2c /user/hadoop/Proj1/data/AccessLog.csv /user/hadoop/Proj1/javacode_c1

Without combiner:
Result:



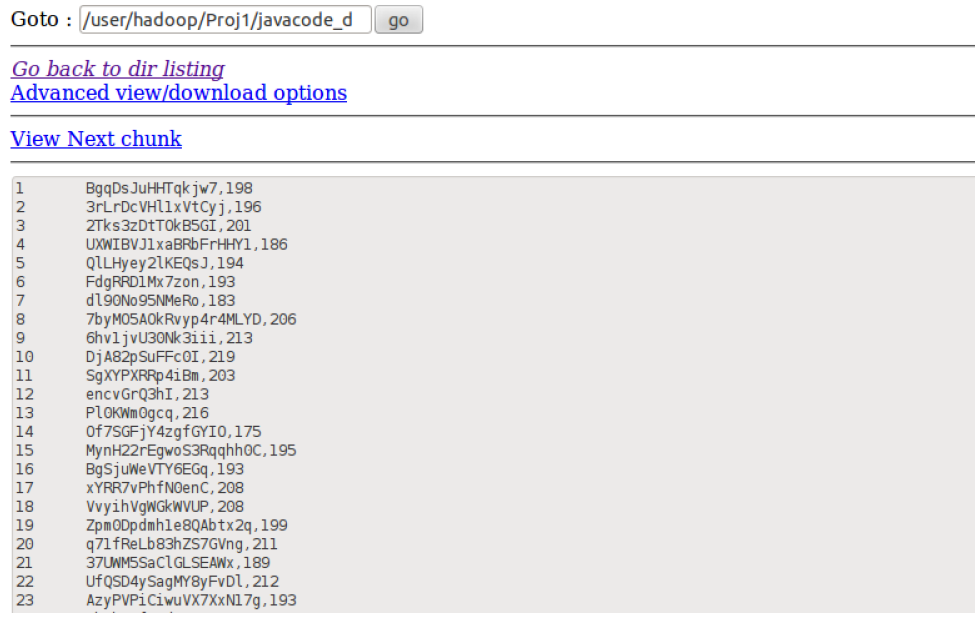
For this task, it used 38760ms.

```
17/09/22 14:16:42 INFO mapred.JobClient:   File Input Format Counters
17/09/22 14:16:42 INFO mapred.JobClient:     Bytes Read=604012544
17/09/22 14:16:42 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 14:16:42 INFO mapred.JobClient:     Map output materialized bytes=96554
364
17/09/22 14:16:42 INFO mapred.JobClient:     Map input records=9655431
17/09/22 14:16:42 INFO mapred.JobClient:     Reduce shuffle bytes=96554364
17/09/22 14:16:42 INFO mapred.JobClient:     Spilled Records=28966293
17/09/22 14:16:42 INFO mapred.JobClient:     Map output bytes=77243448
17/09/22 14:16:42 INFO mapred.JobClient:     Total committed heap usage (bytes)=
1590714368
17/09/22 14:16:42 INFO mapred.JobClient:     CPU time spent (ms)=38760
17/09/22 14:16:42 INFO mapred.JobClient:     Combine input records=0
17/09/22 14:16:42 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1107
17/09/22 14:16:42 INFO mapred.JobClient:     Reduce input records=9655431
17/09/22 14:16:42 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 14:16:42 INFO mapred.JobClient:     Combine output records=0
17/09/22 14:16:42 INFO mapred.JobClient:     Physical memory (bytes) snapshot=18
08633856
```

Task3d.)
We use two mappers. One is to read MyPage line by line, generation key(ID)-value(Name) pairs with a file flag"M". The other one is to read Friends dateset line by line, generating key(MyFriend)-value(1) pairs with a file flag"F". And one reducer takes the output from these mappers(join these two dataset by person id) to count the number of people listing this user as a friend.
We cannot use combiner, because it is not a sum up job. For MyPage dataset, we need to get Name, while for Friends we count the total number. Because when the node is busy, they will skip combiner.

-- sample execution:
> hadoop jar ./project1.jar my.twod_nocombiner /user/hadoop/Proj1/data/MyPage.csv /user/hadoop/Proj1/data/Friends.csv /user/hadoop/Proj1/javacode_d

Without combiner:
Partial result:



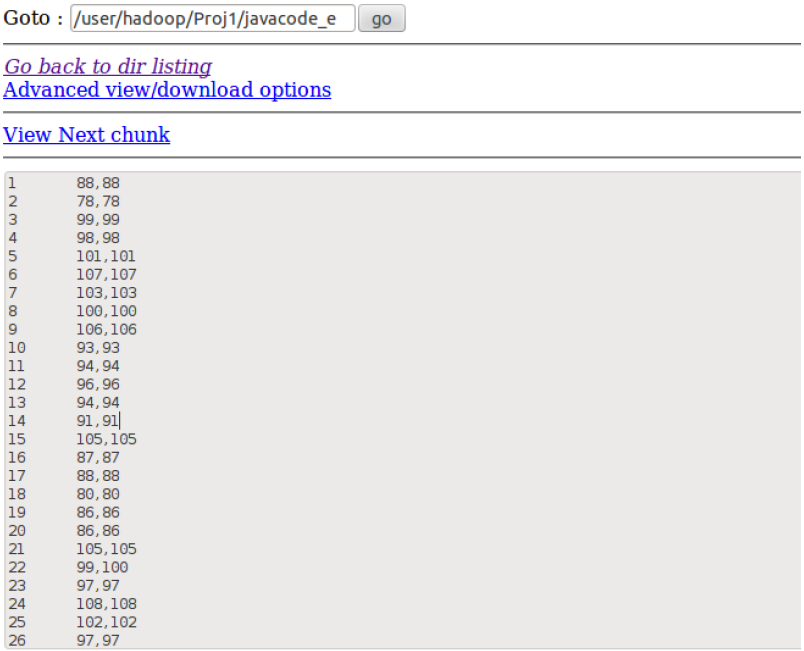
For this task, used 251710ms.

```
17/09/22 12:00:05 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 12:00:05 INFO mapred.JobClient:     Map output materialized bytes=28280
0325
17/09/22 12:00:05 INFO mapred.JobClient:     Map input records=20100000
17/09/22 12:00:05 INFO mapred.JobClient:     Reduce shuffle bytes=282800325
17/09/22 12:00:05 INFO mapred.JobClient:     Spilled Records=60200000
17/09/22 12:00:05 INFO mapred.JobClient:     Map output bytes=242600199
17/09/22 12:00:05 INFO mapred.JobClient:     Total committed heap usage (bytes)=
3533983744
17/09/22 12:00:05 INFO mapred.JobClient:     CPU time spent (ms)=251710
17/09/22 12:00:05 INFO mapred.JobClient:     Combine input records=0
17/09/22 12:00:05 INFO mapred.JobClient:     SPLIT_RAW_BYTES=5438
17/09/22 12:00:05 INFO mapred.JobClient:     Reduce input records=20100000
17/09/22 12:00:05 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 12:00:05 INFO mapred.JobClient:     Combine output records=0
17/09/22 12:00:05 INFO mapred.JobClient:     Physical memory (bytes) snapshot=41
93480704
17/09/22 12:00:05 INFO mapred.JobClient:     Reduce output records=100000
17/09/22 12:00:05 INFO mapred.JobClient:     Virtual memory (bytes) snapshot=765
2909056
17/09/22 12:00:05 INFO mapred.JobClient:     Map output records=20100000
```

Task3e.)
We use one mapper to read AccessLog dataset line by line, generating key(ByWho)-value(WhatPage) pairs. And one reducer takes these output, creating a hashset to store unique WhatPage and a int value to store total number of WhatPage for each key.
We cannot use combiner, because we have to find the unique value.

-- sample execution:
> hadoop jar ./project1.jar my.twoe_nocombiner /user/hadoop/Proj1/data/AccessLog.csv /user/hadoop/Proj1/javacode_e

Without combiner:
Partial result:



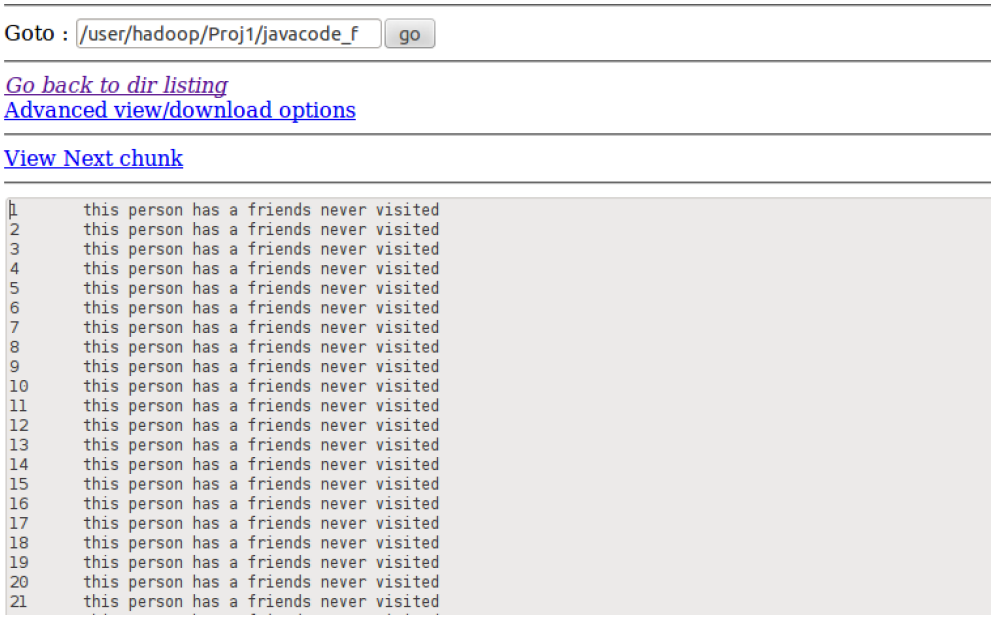
For this task, it used 51460ms.

```
17/09/22 12:30:31 INFO mapred.JobClient:   File Input Format Counters
17/09/22 12:30:31 INFO mapred.JobClient:     Bytes Read=604012544
17/09/22 12:30:31 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 12:30:31 INFO mapred.JobClient:     Map output materialized bytes=15341
4904
17/09/22 12:30:31 INFO mapred.JobClient:     Map input records=9655431
17/09/22 12:30:31 INFO mapred.JobClient:     Reduce shuffle bytes=153414904
17/09/22 12:30:31 INFO mapred.JobClient:     Spilled Records=28966293
17/09/22 12:30:31 INFO mapred.JobClient:     Map output bytes=134103988
17/09/22 12:30:31 INFO mapred.JobClient:     Total committed heap usage (bytes)=
1215594496
17/09/22 12:30:31 INFO mapred.JobClient:     CPU time spent (ms)=51460
17/09/22 12:30:31 INFO mapred.JobClient:     Combine input records=0
17/09/22 12:30:31 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1107
17/09/22 12:30:31 INFO mapred.JobClient:     Reduce input records=9655431
17/09/22 12:30:31 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 12:30:31 INFO mapred.JobClient:     Combine output records=0
17/09/22 12:30:31 INFO mapred.JobClient:     Physical memory (bytes) snapshot=15
22278400
17/09/22 12:30:31 INFO mapred.JobClient:     Reduce output records=100000
```

Task3f.)
We use two mappers. One is to read Friends dataset line by line, generation key(PersonID)-value(MyFriend) pairs with a file flag “F”. The other one is to read AccessLog dataset line by line, generating key(ByWho)-value(WhatPage) pairs with a file flag “A”. And one reducer takes all the output, grouping by PersonID, creating two hashsets to store unique friend id(MyFriend) and page id(WhatPage). If there is a friend of a certain user never appears in the WhatPage set, then we can specify this user declared someone as his friend yet never accessed his friend’s page.
We cannot use combiner, because this is not a sum up job. We have to look at all the mappers not only the local one.

-- sample execution:
> hadoop jar ./project1.jar my.twof_nocombiner /user/hadoop/Proj1/data/Friends.csv /user/hadoop/Proj1/data/AccessLog.csv /user/hadoop/Proj1/javacode_f

Without combiner:
Partial result:



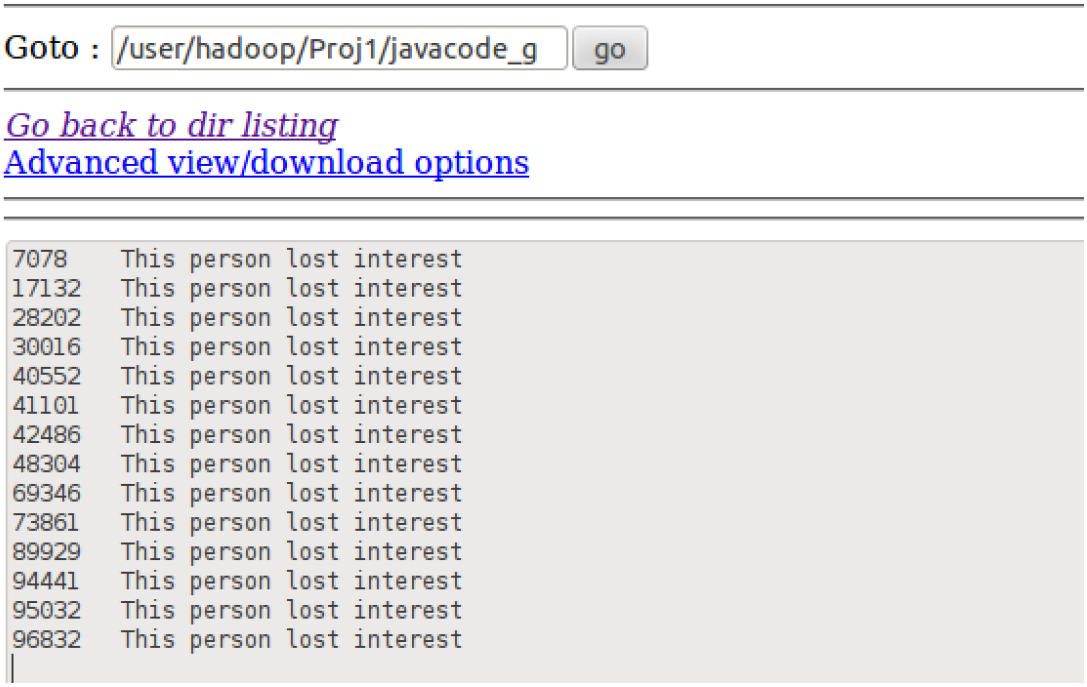
For this task, it used 178080ms.

```
17/09/22 12:39:14 INFO mapred.JobClient:   File Input Format Counters
17/09/22 12:39:14 INFO mapred.JobClient:     Bytes Read=0
17/09/22 12:39:14 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 12:39:14 INFO mapred.JobClient:     Map output materialized bytes=53050
6902
17/09/22 12:39:14 INFO mapred.JobClient:     Map input records=29655431
17/09/22 12:39:14 INFO mapred.JobClient:     Reduce shuffle bytes=530506902
17/09/22 12:39:14 INFO mapred.JobClient:     Spilled Records=88966293
17/09/22 12:39:14 INFO mapred.JobClient:     Map output bytes=471195866
17/09/22 12:39:14 INFO mapred.JobClient:     Total committed heap usage (bytes)=
4775268352
17/09/22 12:39:14 INFO mapred.JobClient:     CPU time spent (ms)=178080
17/09/22 12:39:14 INFO mapred.JobClient:     Combine input records=0
17/09/22 12:39:14 INFO mapred.JobClient:     SPLIT_RAW_BYTES=7529
17/09/22 12:39:14 INFO mapred.JobClient:     Reduce input records=29655431
17/09/22 12:39:14 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 12:39:14 INFO mapred.JobClient:     Combine output records=0
17/09/22 12:39:14 INFO mapred.JobClient:     Physical memory (bytes) snapshot=56
73529344
17/09/22 12:39:14 INFO mapred.JobClient:     Reduce output records=100000
```

Task3g.)
We use one mapper to read AccessLog dateset line by line, generating key(ByWho)-value(AccessTime) pairs. And one reducer takes the output, grouping by ByWho. In the reducer, we find the last visit time(maximum number in AccessTime) and initialize a time unit, finding out whoes last visit is before that day.
We cannot use combiner, we have to look at all the mappers to find the global maximum.

-- sample execution:
> hadoop jar ./project1.jar my.twog_nocombiner /user/hadoop/Proj1/data/AccessLog.csv /user/hadoop/Proj1/javacode_g

Without combiner:
Result:



For this task, it used 49470ms.

```
17/09/22 12:51:20 INFO mapred.JobClient:   File Input Format Counters
17/09/22 12:51:20 INFO mapred.JobClient:     Bytes Read=604012544
17/09/22 12:51:20 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 12:51:20 INFO mapred.JobClient:     Map output materialized bytes=16306
9936
17/09/22 12:51:20 INFO mapred.JobClient:     Map input records=9655431
17/09/22 12:51:20 INFO mapred.JobClient:     Reduce shuffle bytes=163069936
17/09/22 12:51:20 INFO mapred.JobClient:     Spilled Records=28966293
17/09/22 12:51:20 INFO mapred.JobClient:     Map output bytes=143759020
17/09/22 12:51:20 INFO mapred.JobClient:     Total committed heap usage (bytes)=
1225908224
17/09/22 12:51:20 INFO mapred.JobClient:     CPU time spent (ms)=49470
17/09/22 12:51:20 INFO mapred.JobClient:     Combine input records=0
17/09/22 12:51:20 INFO mapred.JobClient:     SPLIT_RAW_BYTES=1107
17/09/22 12:51:20 INFO mapred.JobClient:     Reduce input records=9655431
17/09/22 12:51:20 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 12:51:20 INFO mapred.JobClient:     Combine output records=0
17/09/22 12:51:20 INFO mapred.JobClient:     Physical memory (bytes) snapshot=15
21668096
17/09/22 12:51:20 INFO mapred.JobClient:     Reduce output records=14
```

Task3h.)
We use one mapper to read Friends dataset line by line, generating key(PersonID)-value(1) pairs. And one reducer gets average number of friends for all the users and that user's actual number of friends by reading in mapper output. We create a hashtable to store key(personID)-value(number of friends).
We cannot use combiner, because some mapper will skip combiner. In this way, the total number of friends for all user and the number of user will be wrong.

-- sample execution:
> hadoop jar ./project1.jar my.twoh /user/hadoop/Proj1/data/Friends.csv /user/hadoop/Proj1/javacode_h1
Partial Report:

Goto : /user/hadoop/Proj1/javacode_h1 go

Go back to dir listing
Advanced view/download options

View Next chunk

```
99999   This person is popular
99998   This person is popular
99997   This person is popular
99996   This person is popular
99994   This person is popular
99993   This person is popular
99992   This person is popular
99991   This person is popular
99990   This person is popular
99989   This person is popular
99988   This person is popular
99986   This person is popular
99982   This person is popular
99976   This person is popular
99974   This person is popular
99973   This person is popular
99970   This person is popular
99969   This person is popular
```

For this task, it used 103120ms.

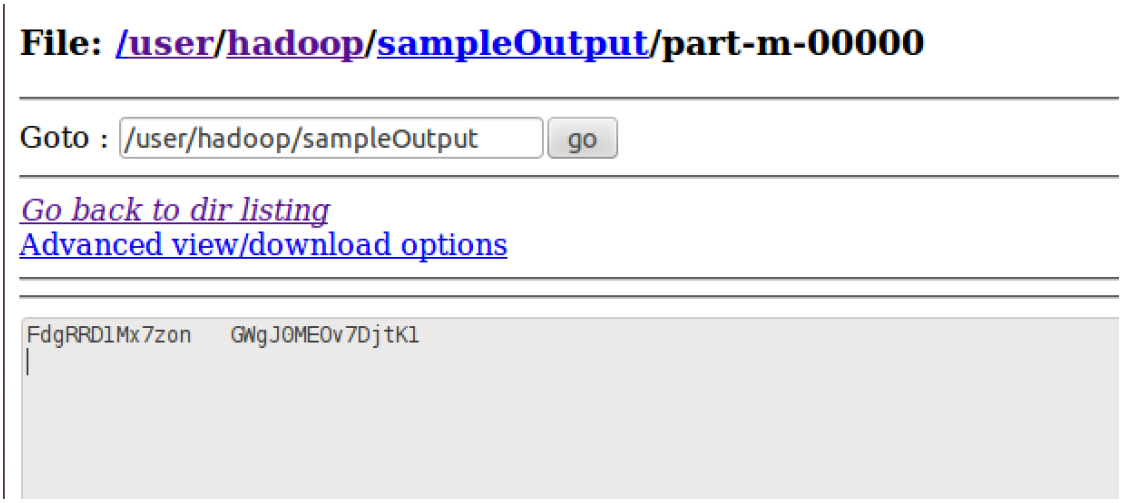
```
17/09/22 15:39:37 INFO mapred.JobClient:   File Input Format Counters
17/09/22 15:39:37 INFO mapred.JobClient:     Bytes Read=1312307332
17/09/22 15:39:37 INFO mapred.JobClient:   Map-Reduce Framework
17/09/22 15:39:37 INFO mapred.JobClient:     Map output materialized bytes=26000
0120
17/09/22 15:39:37 INFO mapred.JobClient:     Map input records=20000000
17/09/22 15:39:37 INFO mapred.JobClient:     Reduce shuffle bytes=260000120
17/09/22 15:39:37 INFO mapred.JobClient:     Spilled Records=60000000
17/09/22 15:39:37 INFO mapred.JobClient:     Map output bytes=220000000
17/09/22 15:39:37 INFO mapred.JobClient:     Total committed heap usage (bytes)=
2521513984
17/09/22 15:39:37 INFO mapred.JobClient:     CPU time spent (ms)=103120
17/09/22 15:39:37 INFO mapred.JobClient:     Combine input records=0
17/09/22 15:39:37 INFO mapred.JobClient:     SPLIT_RAW_BYTES=2420
17/09/22 15:39:37 INFO mapred.JobClient:     Reduce input records=20000000
17/09/22 15:39:37 INFO mapred.JobClient:     Reduce input groups=100000
17/09/22 15:39:37 INFO mapred.JobClient:     Combine output records=0
17/09/22 15:39:37 INFO mapred.JobClient:     Physical memory (bytes) snapshot=31
50733312
```

4. Accomplishing Analytics Tasks Apache Pig

task4a.)
Loads in your generated MyPage data and first filters only the natiionality specified. While filtering, it only stores name and hobby to then write back to an output file.

-- sample execution:
>pig -f task4a.pig -param MyPage=/user/hadoop/Proj1/data/MyPage.csv -param nationality=LDPzI2fYhucyXJ -param output=sampleOutput

Result:



For task a, used 14 second.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-21 23:54:55     2017-09-21 23:55:09     FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime       A
lias    Feature Outputs
job_201709212003_0030   1       0       2       2       2       2       0       0
0       0       data,name_hobby,nat_filt        MAP_ONLY        hdfs://localhost
:8020/user/hadoop/sampleOutput,

Input(s):
Successfully read 100000 records (5599980 bytes) from: "/user/hadoop/Proj1/data/
MyPage.csv"

Output(s):
Successfully stored 1 records (30 bytes) in: "hdfs://localhost:8020/user/hadoop/
sampleOutput"
```

task4b.)
Loads in your generated MyPage data and groups all data by country code while counting the number of pages (rows) that have that given country code.

-- sample execution:
> pig -f task2b.pig -param MyPage=/user/hadoop/Proj1/data/MyPage.csv -param output=pig_task4b

Result:

File: /user/hadoop/pig_task4b/part-r-00000

Goto : /user/hadoop/pig_task4b go

Go back to dir listing
Advanced view/download options

```
1       9860
2       10026
3       10058
4       10117
5       10110
6       10083
7       9919
8       9815
9       10042
10      9970
```

For taskb, it used 19 seconds.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 00:12:47     2017-09-22 00:13:06     GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709212003_0031   1       1       2       2       2       2       8       8
8       8       data,nat_cit,nat_group  GROUP_BY,COMBINER       hdfs://localhost
:8020/user/hadoop/pig_task4b,

Input(s):
Successfully read 100000 records (5599980 bytes) from: "/user/hadoop/Proj1/data/
MyPage.csv"

Output(s):
Successfully stored 10 records (77 bytes) in: "hdfs://localhost:8020/user/hadoop
/pig_task4b"
```

task4c.)
Loads AccessLog data and groups data based on whatpage visited. It counts the number of times each unique id was viewed, sorts the counts in descending order, then takes the top 10.

-- sample execution:
>pig -f task4c.pig -param AccessLog=/pigdata/sampleAccessLog.csv -param output=sampleOutput

Result:

File: /user/hadoop/pig_task4c/part-r-00000

Goto : /user/hadoop/pig_task4c go

Go back to dir listing
Advanced view/download options

```
59666   137
37883   137
24271   137
38048   136
62485   136
63713   136
38339   136
33985   136
54706   135
53314   135
```

For this task, it used 187 seconds.

```
1.2.1   0.13.0  hadoop  2017-09-22 00:29:32     2017-09-22 00:32:39     GROUP_BY
,ORDER_BY,LIMIT

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709212003_0036   9       1       27      12      23      23      92      9
2       92      92      counted_log,data,grouped_log    GROUP_BY,COMBINER
job_201709212003_0037   1       1       1       1       1       1       8       8
8       8       ordered_log     SAMPLER
job_201709212003_0038   1       1       1       1       1       1       8       8
8       8       ordered_log     ORDER_BY,COMBINER
job_201709212003_0039   1       1       1       1       1       1       8       8
8       8       ordered_log             hdfs://localhost:8020/user/hadoop/pig_ta
sk4c,

Input(s):
Successfully read 9655431 records (604015937 bytes) from: "/user/hadoop/Proj1/da
ta/AccessLog.csv"

Output(s):
Successfully stored 10 records (100 bytes) in: "hdfs://localhost:8020/user/hadoo
p/pig_task4c"
```

In Hadoop, PersonID is not ordered, so the results may be different.

Task4d.)
Loads in both mypage and friends data. Groups friend data by myfriend (ids that are the friend of someone else) and counts the number of instances. Joins the mypage and newly grouped datasets on id, and then takes the name and count columns to report.

-- sample execution:
> pig task4d.pig

Partial Result:

File: /user/hadoop/pig_task4d.out/part-r-00000

Goto : /user/hadoop/pig_task4d.out go

Go back to dir listing
Advanced view/download options

View Next chunk

```
1       BgqDsJuHHTqkjw7 198
2       3rLrDcVHllxVtCyj        196
3       2Tks3zDtTOkB5GI 201
4       UXWIBVJlxaBRbFrHHY1     186
5       QlLHyey2lKEQsJ  194
6       FdgRRDlMx7zon   193
7       dl9ONo95NMeRo   183
8       7byMO5AOkRvyp4r4MLYD    206
9       6hvljvU30Nk3iii 213
10      DjA82pSuFFc0I   219
11      SgXYPXRRp4iBm   203
12      encvGrQ3hI      213
13      Pl0KWm0gcq      216
14      Of7SGFjY4zgfGYIO        175
15      MynH22rEgwoS3RqqhhOC    195
16      BgSjuWeVTY6EGq  193
17      xYRR7vPhfNOenC  208
18      VvyihVgWGkWVUP  208
19      ZpmODpdmhle8QAbtx2q     199
```

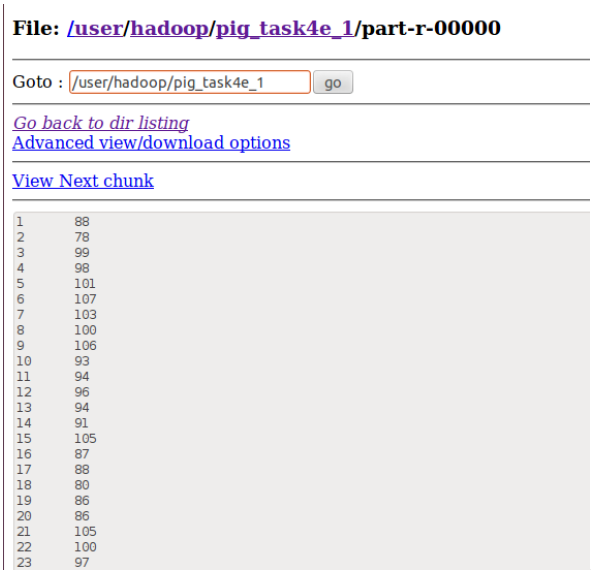
For task d, it used 279 seconds.

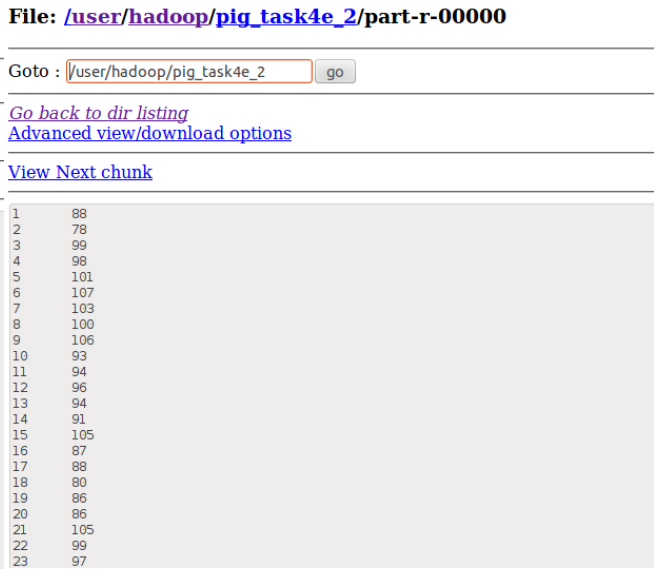
```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 01:22:39     2017-09-22 01:27:18     HASH_JOI
N,GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709212003_0047   20      2       25      15      23      23      219     2
19      219     219     countedFriends,friends,groupedFriends   GROUP_BY,COMBINE
R
job_201709212003_0048   2       1       4       3       3       3       9       9
9       9       joinFL,myPage,selectedFL        HASH_JOIN       hdfs://localhost
:8020/user/hadoop/pig_task4d.out,

Input(s):
Successfully read 20000000 records (1312314832 bytes) from: "/user/hadoop/Proj1/
data/Friends.csv"
Successfully read 100000 records from: "/user/hadoop/Proj1/data/MyPage.csv"

Output(s):
Successfully stored 100000 records (2589094 bytes) in: "hdfs://localhost:8020/us
er/hadoop/pig_task4d.out"
```

task4e.)
Load accesslog and group by bywho id. Count the number of accesses by each id. Then find the unique pages and the number of accesses. Join the tables of unique and number of accesses.

-- sample execution:
> pig -f task4e.pig -param AccessLog=/user/hadoop/Proj1/data/AccessLog.csv -param output1=pig_task4e_1 -param output2=pig_task4e_2

Partial Result: pig_task4e_1: total, pig_task4e_2: distinct





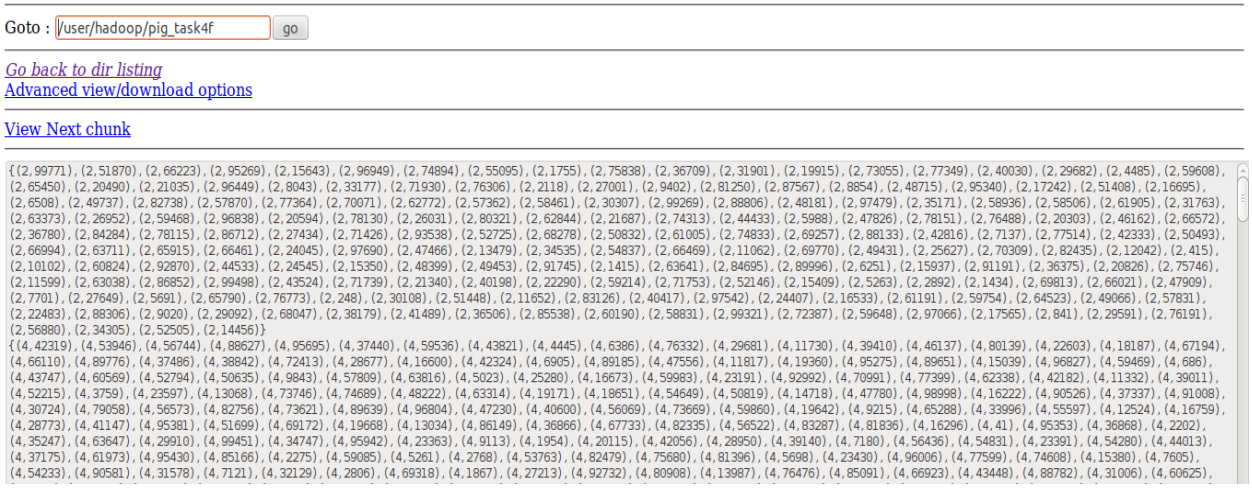
For this task, it used 180 seconds.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 01:46:51     2017-09-22 01:49:51     GROUP_BY

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709212003_0051   9       1       24      11      21      22      146     1
46      146     146     1-1,access_log,count_distinct_log,counted_log,grouped_lo
g,uniqe_page    GROUP_BY,MULTI_QUERY    hdfs://localhost:8020/user/hadoop/pig_ta
sk4e_1,hdfs://localhost:8020/user/hadoop/pig_task4e_2,

Input(s):
Successfully read 9655431 records (604015937 bytes) from: "/user/hadoop/Proj1/da
ta/AccessLog.csv"

Output(s):
Successfully stored 100000 records (926504 bytes) in: "hdfs://localhost:8020/use
r/hadoop/pig_task4e_1"
Successfully stored 100000 records (926318 bytes) in: "hdfs://localhost:8020/use
r/hadoop/pig_task4e_2"
```

task4f.)
Due to people can check their friends' page many time, so first find unique accesslog of ByWho and WhatPage, and then, find the difference between friends dataset and this unique log dataset.

-- sample execution:
> pig -f task4f.pig -param Friends=/user/hadoop/Proj1/data/Friends.csv -param AccessLog=/user/hadoop/Proj1/data/AccessLog.csv -param output=pig_task4f

Partial Result: The first element of tuple is the person id, the second element is the id of friend



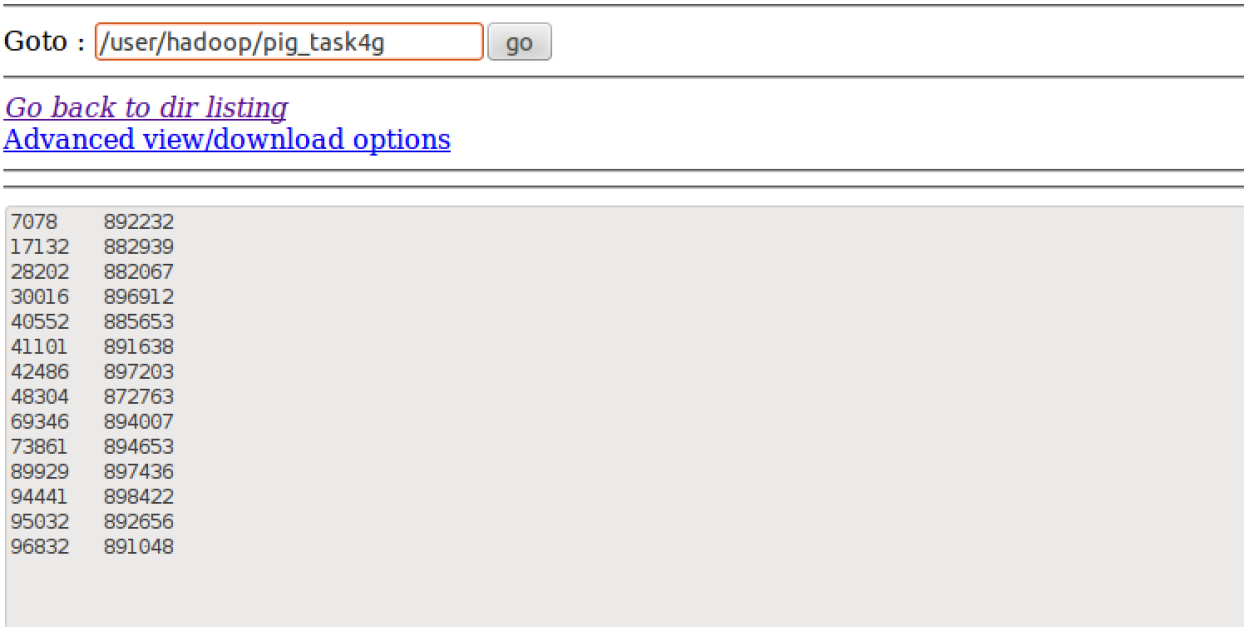
For this task, it used 451 seconds.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 02:06:47     2017-09-22 02:14:18     COGROUP,
DISTINCT

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709212003_0054   9       1       28      11      21      20      118     1
18      118     118     accessLog,sAccessLog    DISTINCT
job_201709212003_0055   22      2       42      9       19      16      270     2
70      270     270     cogroup_data,friends,sFriends,sub_data  COGROUP hdfs://l
ocalhost:8020/user/hadoop/pig_task4f,

Input(s):
Successfully read 9655431 records (604015937 bytes) from: "/user/hadoop/Proj1/da
ta/AccessLog.csv"
Successfully read 20000000 records from: "/user/hadoop/Proj1/data/Friends.csv"

Output(s):
Successfully stored 100000 records (275216211 bytes) in: "hdfs://localhost:8020/
user/hadoop/pig_task4f"
```

Task4g.)
Load in the access log data. Group the logs bywho id and compute the max accesstime for each id (this is the last time the person accessed facebook). Filter all those that have less than some time unit, ie havent been on facebook past that date.

-- sample execution:
> pig -f task4g.pig -param AccessLog=/user/hadoop/Proj1/data/AccessLog.csv -param time_unit=900000 -param output=pig_task4g

Partial Result:



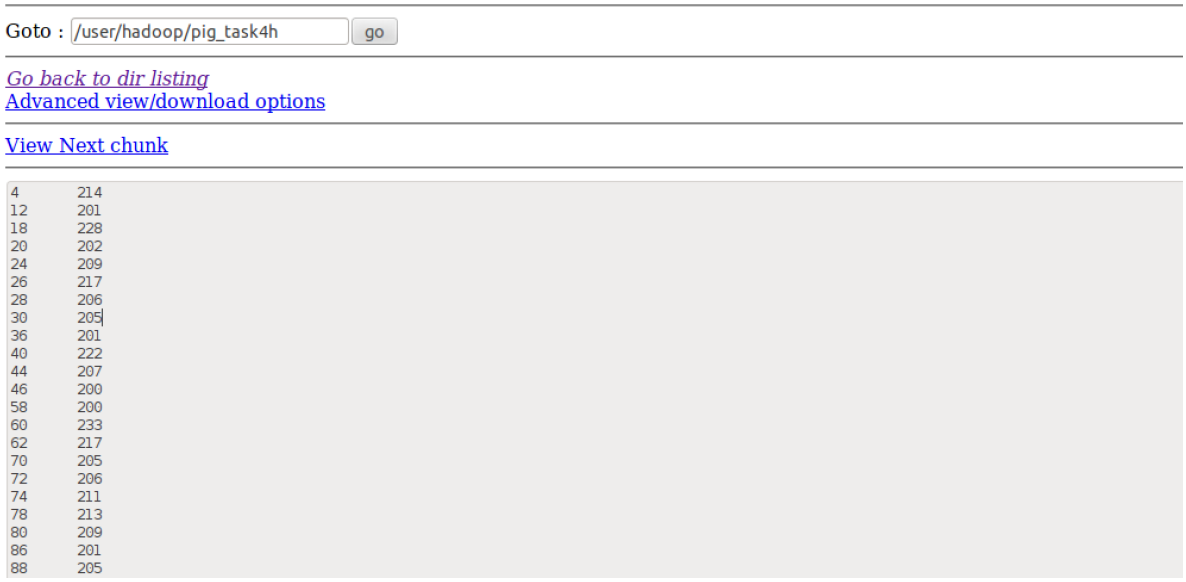
For this task, it used 120 seconds.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 11:01:23     2017-09-22 11:03:23     GROUP_BY
,FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime        A
lias    Feature Outputs
job_201709221039_0004   9       1       23      11      21      22      86      8
6       86      86      access_log,filtered_log,grouped_log,max_log     GROUP_BY
,COMBINER       hdfs://localhost:8020/user/hadoop/pig_task4g,

Input(s):
Successfully read 9655431 records (604015937 bytes) from: "/user/hadoop/Proj1/da
ta/AccessLog.csv"

Output(s):
Successfully stored 14 records (181 bytes) in: "hdfs://localhost:8020/user/hadoo
p/pig_task4g"
```

task4h.)
Load in friends data. Group by id and count the number of friends each person has. Calculated the average by grouping all people into one group, count the total number of friends, and divide by total ids. Then filter all people with more than the average.

-- sample execution:
> pig -f task4h.pig -param Friends=/user/hadoop/Proj1/data/Friends.csv -param output=pig_task4h

Partial Result:



For this task, it used 258 seconds.

```
HadoopVersion   PigVersion      UserId  StartedAt       FinishedAt      Features
1.2.1   0.13.0  hadoop  2017-09-22 02:50:16     2017-09-22 02:54:34     GROUP_BY
,FILTER

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTIme      AvgMapTime      MedianMa
pTime   MaxReduceTime   MinReduceTime   AvgReduceTime   MedianReducetime       A
lias    Feature Outputs
job_201709212003_0059   20      2       22      12      20      21      194     1
94      194     194     counted_friends,data,grouped_friends    GROUP_BY,COMBINE
R
job_201709212003_0060   1       1       1       1       1       1       8       8
8       8       grouped_avg,grouped_counted     GROUP_BY,COMBINER
job_201709212003_0061   1       0       2       2       2       2       0       0
0       0       filtered_friends        MAP_ONLY        hdfs://localhost:8020/us
er/hadoop/pig_task4h,

Input(s):
Successfully read 20000000 records (1312314832 bytes) from: "/user/hadoop/Proj1/
data/Friends.csv"

Output(s):
Successfully stored 50903 records (503327 bytes) in: "hdfs://localhost:8020/user
```

	Mapreduce without combiner	Mapreduce with combiner	Pig
a	0.21s	NA	14s
b	0.69s	0.6s	19s
c	38.76s	NA	187s
d	251.71s	NA	279s
e	51.46s	NA	180s
f	178.08s	NA	451s
g	49.47s	NA	120s
h	103.12s	NA	258s

Based on this table, for Hadoop, not every task could use combiner, but when can use it, the one with combiner will be quicker than the one without. For Pig and Hadoop, Hadoop has a better performance in time than Pig does, especially for small tasks. And for Hadoop, you can write any kind of function as you can, but for Pig, you only can use the built-in functions. Hadoop is more flexible than Pig. For Pig, it is much easier to learn than Hadoop. To choose Hadoop or Pig depends on people's background and the purpose of their tasks.