Jeb Kilfoyle and Damon George
CPEN 435-01

Association Rule Mining Final Project

Intro:

For this project, we wrote a Hadoop MapReduce program to mine the confidence of association rules. From previous homework, we already had access to a MapReduce program that counts the number of files that CIDs and CID pairs appear in, where CID refers to a Concept Identification. The purpose of this project was to build another MapReduce program to parse the output of that first job, and compute the confidence of all associations—an association rule being a X -> Y pair of CIDs. The confidence of such an association can be computed as $conf(X, Y) = \frac{Count(X,Y)}{Count(X)}$, where the counts are provided by the first mapreduce job.

A Hadoop MapReduce program follows a strict structure. The input files are first mapped to key value pairs by the Mapper function. These key value pairs are then sorted, grouped, and sent to the Reducer function so that each Reducer receives all the values for a given key. The Reducer then processes those values and emits the final key value pairs to output files. In addition to writing this MapReduce association rule miner program, this project also entailed extensively tested on Amazon's EMR clusters, which are designed to run Hadoop programs. On the EMR clusters, the execution time of the program was tested for different cluster sizes, for different amounts of input files, and using both the Hadoop file system and the Amazon S3 file system.

Implementation:

Our Hadoop MapReduce implementation consisted of four parts: the first mapping function, the first reduce function, the second mapping function and the second reduce function.

For the first mapping we created an empty arraylist, checked each CID from the file, and added it to the arraylist if it was not already present. Once completed we sorted this list of our CIDs. We then loop over all CID values and all CID pairs, represented as two comma separated values. Our first reduce function then goes through and counts the number of each CID and CID pair as a value associated with each key (CID or CID pair).

Our second map function takes the output of our first MapReduce, CID or CID pairs combined with the number of occurrences, and processes the key part of it. If the input is a pair of CIDs, then the map splits the pair and outputs twice, where the key is each CID. The value is the other CID comma separated with the Count. The map outputs twice to reflect the association in both direction $(X, Y) \Rightarrow (X, Y), (Y, X)$. If the map's input is only a single CID, then the output is one key-value pair where the key is the CID and the value is a blank string comma separated with the Count. The second reducer function starts by initializing a count for keeping track of all X for each (X, Y) pair we see. We create an arraylist to store CID pairs so that we may loop over them multiple times. We then loop through all possible our CID and CID pairs, adding pairs to the arraylist, and looking for a single CID. Once we find a single CID, we assign it as our current value, where current value holds a full pair (X, Y) .When the second CID in a pair is blank, the case of a single CID, we set our count equal to the second value of our input, which was the number of occurrences from the first MapReduce. We then stop our loop and start again, no longer adding to our arraylist. We finish by assigning current value to each new pair and then dividing by a double version of our count, to calculate $conf(X, Y)$. Once we have exhausted the list, we return to our arraylist in case any values were stored before we initialized our current value.

Performance:

To properly test this program, the two MapReduce jobs were timed for 4 different numbers of slave nodes and 4 different numbers of input files. The results of these executions are shown below in Table 1 in the form of <minutes:seconds:milliseconds>.

Table 1: Scalability in terms of Slaves and Input

		250 input files	500 input files	1000 input files	2000 input files
2 Slaves	Job 1	09:02:995	11:40:383	23:05:987	49:12:498
	Job 2	01:06:524	01:09:915	01:24:551	01:58:900
4 Slaves	Job 1	03:20:246	07:07:352	14:10:071	22:47:189
	Job 2	01:03:514	01:03:556	01:16:721	01:24:882
8 Slaves	Job 1	01:53:312	03:29:120	06:00:116	13:56:597
	Job 2	00:48:559	00:53:503	00:56:502	01:04:828
16 Slaves	Job 1	02:18:740	03:41:405	06:46:389	12:31:767
	Job 2	00:51:699	00:52:678	01:01:680	01:13:669

Reading Table 1 from left to right, it is clear that doubling the number of input files approximately double the execution time of the first job, the CID counter from the previous homework. For example, with 4 slaves, the Job 1 execution time doubles from about 3 to 7 minutes, then 7 to 14 minutes, and finally 14 to 23 minutes. This shows that the execution time of Job 1 increases approximately linearly with the increasing number of input files, with a few outliers such as the 250 input files running with 2 Slaves for 9 minutes, which is a little larger than expected.

Reading Table 1 from top to bottom, it is also obvious that the execution times of Job 2 approximately halve with doubling the number of slaves, which supports the expected inverse relationship. This shows that Job 2 scales both with increasing the input and increasing the number of slave nodes. However, increasing the number of slaves makes less difference as the cluster size increases. After a dozen slaves, the increase in performance is not necessarily worth the cost of the extra nodes.

Job 1 is also clearly the limiting factor in the execution times of all the jobs in Table 1. Job 2 is much faster than Job 1, requiring around a minute to execute for almost every trial, and only increases by seconds for increasing the input. With 2 Slaves, the execution time only doubled from 250 input files to 2000. With 8 Slaves, the time only increased by about 15 seconds from 250 to 2000 input files. This shows that the MapReduce program written for calculating confidence is very efficient and scalable for different numbers of input files. The execution of the second job also barely changes when reading the table from top to bottom. The total range of times is only about 1 second, which shows that increasing the size of the Hadoop cluster only makes a small impact on the execution of the second job.

It is also interesting to note the trials using 16 Slaves, which are actually slightly longer than the 8 Slave trials. This is likely due to the fact that, because of technical difficulties, the 16 Slave cluster used the AWS m4.large instances instead of the m4.xlarge instances that the rest of the trials used. This demonstrates how much of an impact the type of AWS instance affects the execution—16 m4.large instances were still slower than only 8 of the m4.xlarge instances.

In addition to testing the program for different cluster and input sizes, the Association mining was also tested when logging to the AWS S3 storage. All the previous trials from Table 1 used the Hadoop File System (HDFS), and those trials for the 8 Slave Cluster were copied below into Table 2 to compare against the execution times for an 8 Slave Cluster using S3 storage.

Table 2: Logging in HDFS vs AWS S3

		250 input files	500 input files	1000 input files	2000 input files
Using HDFS (8 Slaves)	Job 1	01:53:312	03:29:120	06:00:116	13:56:597
	Job 2	00:48:559	00:53:503	00:56:502	01:04:828
Using S3 (8 Slaves)	Job 1	05:16:086	11:01:340	17:14:433	36:21:558
	Job 2	01:26:357	01:28:429	01:43:232	01:52:519

By inspecting Table 2, using the remote S3 storage for input and output of the Hadoop program more than doubled the execution times for Job 1. The trials for Job 2 also increased, but Job 2 is so fast that it hardly compares to the time that Job 1 takes. These trials demonstrate that running Hadoop programs on AWS EMR clusters with S3 storage greatly impacts performance, even though using S3 would be very useful with large datasets.

Conclusion:

Overall, programming the association rule mining Hadoop program was not overly difficult. The second job, which was coded for this project, efficiently scaled both with increasing input and cluster size. The first job, from the previous homework, was the limiting factor in execution time. This is likely because the first job must parse the large input files and also must sort the input in its map function in order to effectively count the CID pairs. It was also interesting to note the impact that using a different instance type has on execution time.

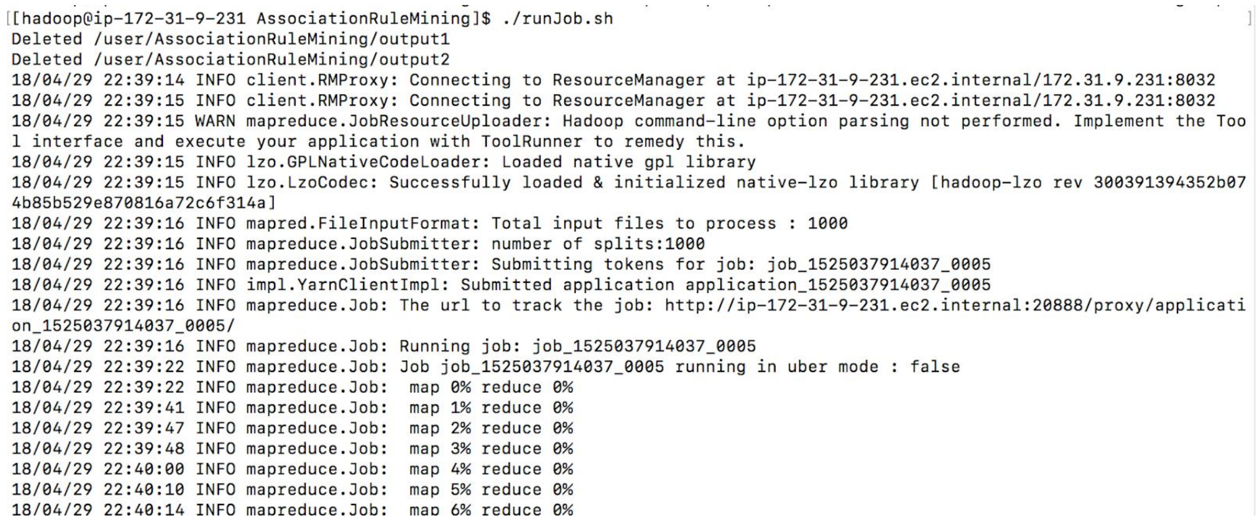
Furthermore, the program ran significantly slower using the S3 storage, which likely has a big impact on companies who process massive datasets on AWS from S3 storage.

Contributions:

Both Damon and Jeb coded the assignment together. Damon tested the performance of the program and therefore wrote the performance, and intro, sections of the report. Jeb created the power point presentation and finished the rest of the report.

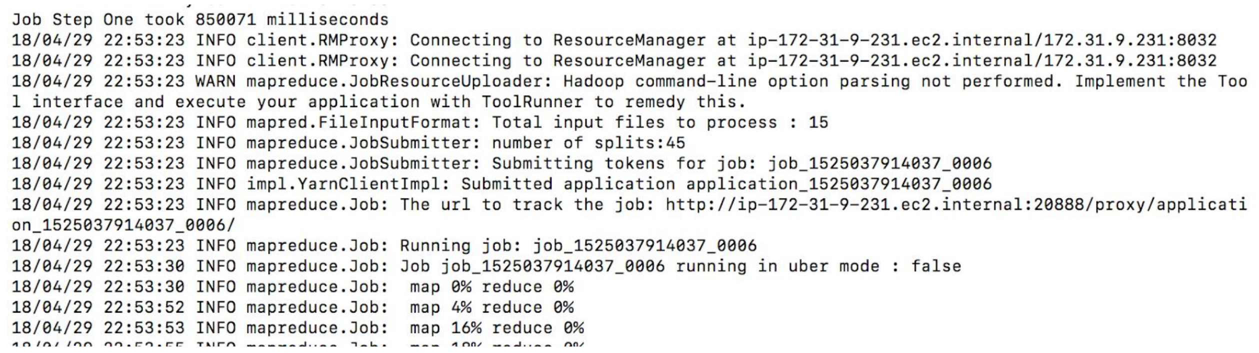
Screenshots:

Starting the Program



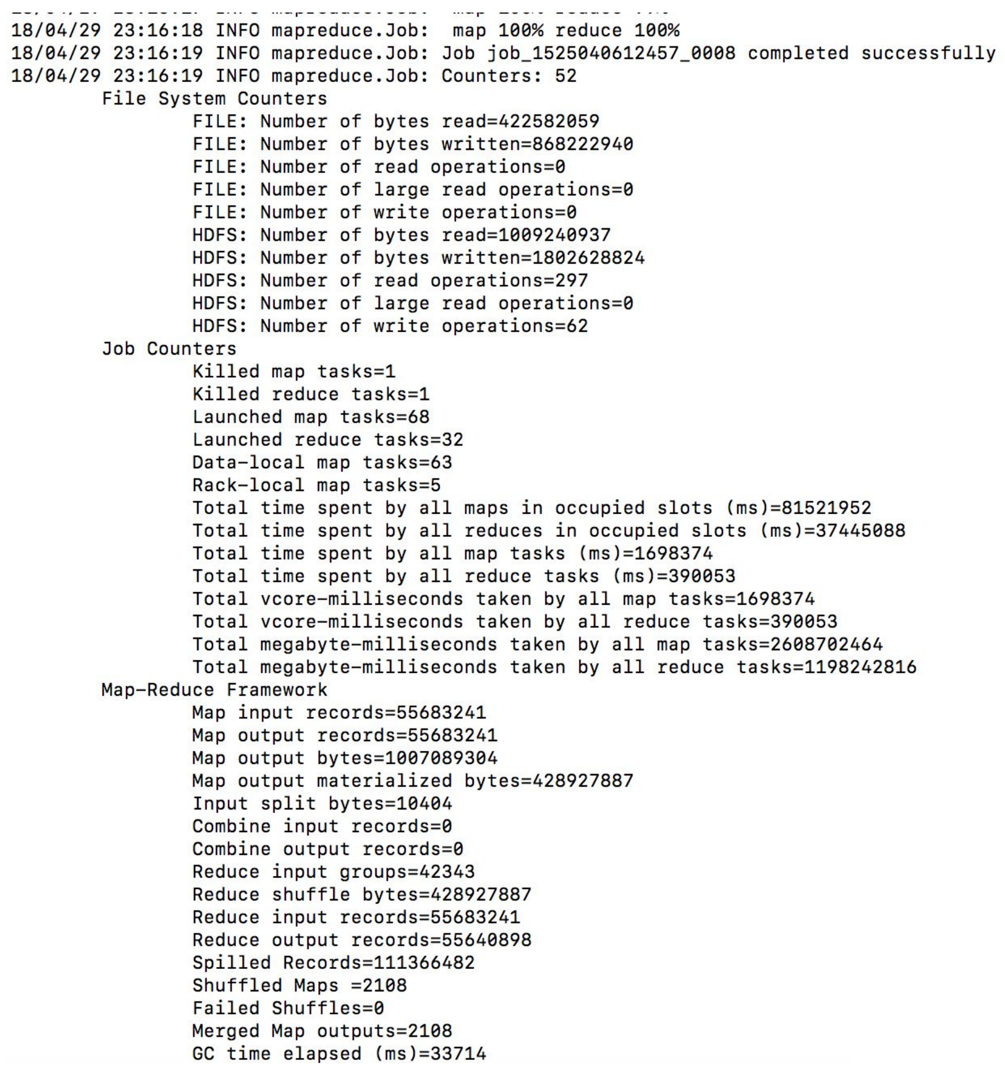
```
[[hadoop@ip-172-31-9-231 AssociationRuleMining]$ ./runJob.sh
Deleted /user/AssociationRuleMining/output1
Deleted /user/AssociationRuleMining/output2
18/04/29 22:39:14 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-9-231.ec2.internal/172.31.9.231:8032
18/04/29 22:39:15 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-9-231.ec2.internal/172.31.9.231:8032
18/04/29 22:39:15 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Too
l interface and execute your application with ToolRunner to remedy this.
18/04/29 22:39:15 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
18/04/29 22:39:15 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 300391394352b07
4b85b529e870816a72c6f314a]
18/04/29 22:39:16 INFO mapred.FileInputFormat: Total input files to process : 1000
18/04/29 22:39:16 INFO mapreduce.JobSubmitter: number of splits:1000
18/04/29 22:39:16 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1525037914037_0005
18/04/29 22:39:16 INFO impl.YarnClientImpl: Submitted application application_1525037914037_0005
18/04/29 22:39:16 INFO mapreduce.Job: The url to track the job: http://ip-172-31-9-231.ec2.internal:20888/proxy/applicati
on_1525037914037_0005/
18/04/29 22:39:16 INFO mapreduce.Job: Running job: job_1525037914037_0005
18/04/29 22:39:22 INFO mapreduce.Job: Job job_1525037914037_0005 running in uber mode : false
18/04/29 22:39:22 INFO mapreduce.Job:  map 0% reduce 0%
18/04/29 22:39:41 INFO mapreduce.Job:  map 1% reduce 0%
18/04/29 22:39:47 INFO mapreduce.Job:  map 2% reduce 0%
18/04/29 22:39:48 INFO mapreduce.Job:  map 3% reduce 0%
18/04/29 22:40:00 INFO mapreduce.Job:  map 4% reduce 0%
18/04/29 22:40:10 INFO mapreduce.Job:  map 5% reduce 0%
18/04/29 22:40:14 INFO mapreduce.Job:  map 6% reduce 0%
```

Job 1 finishing and Job 2 starting



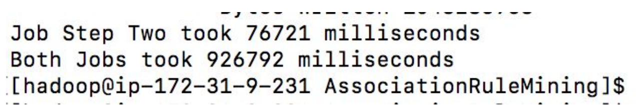
```
Job Step One took 850071 milliseconds
18/04/29 22:53:23 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-9-231.ec2.internal/172.31.9.231:8032
18/04/29 22:53:23 INFO client.RMProxy: Connecting to ResourceManager at ip-172-31-9-231.ec2.internal/172.31.9.231:8032
18/04/29 22:53:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Too
l interface and execute your application with ToolRunner to remedy this.
18/04/29 22:53:23 INFO mapred.FileInputFormat: Total input files to process : 15
18/04/29 22:53:23 INFO mapreduce.JobSubmitter: number of splits:45
18/04/29 22:53:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1525037914037_0006
18/04/29 22:53:23 INFO impl.YarnClientImpl: Submitted application application_1525037914037_0006
18/04/29 22:53:23 INFO mapreduce.Job: The url to track the job: http://ip-172-31-9-231.ec2.internal:20888/proxy/applicati
on_1525037914037_0006/
18/04/29 22:53:23 INFO mapreduce.Job: Running job: job_1525037914037_0006
18/04/29 22:53:30 INFO mapreduce.Job: Job job_1525037914037_0006 running in uber mode : false
18/04/29 22:53:30 INFO mapreduce.Job:  map 0% reduce 0%
18/04/29 22:53:52 INFO mapreduce.Job:  map 4% reduce 0%
18/04/29 22:53:53 INFO mapreduce.Job:  map 16% reduce 0%
```

Both Jobs are done



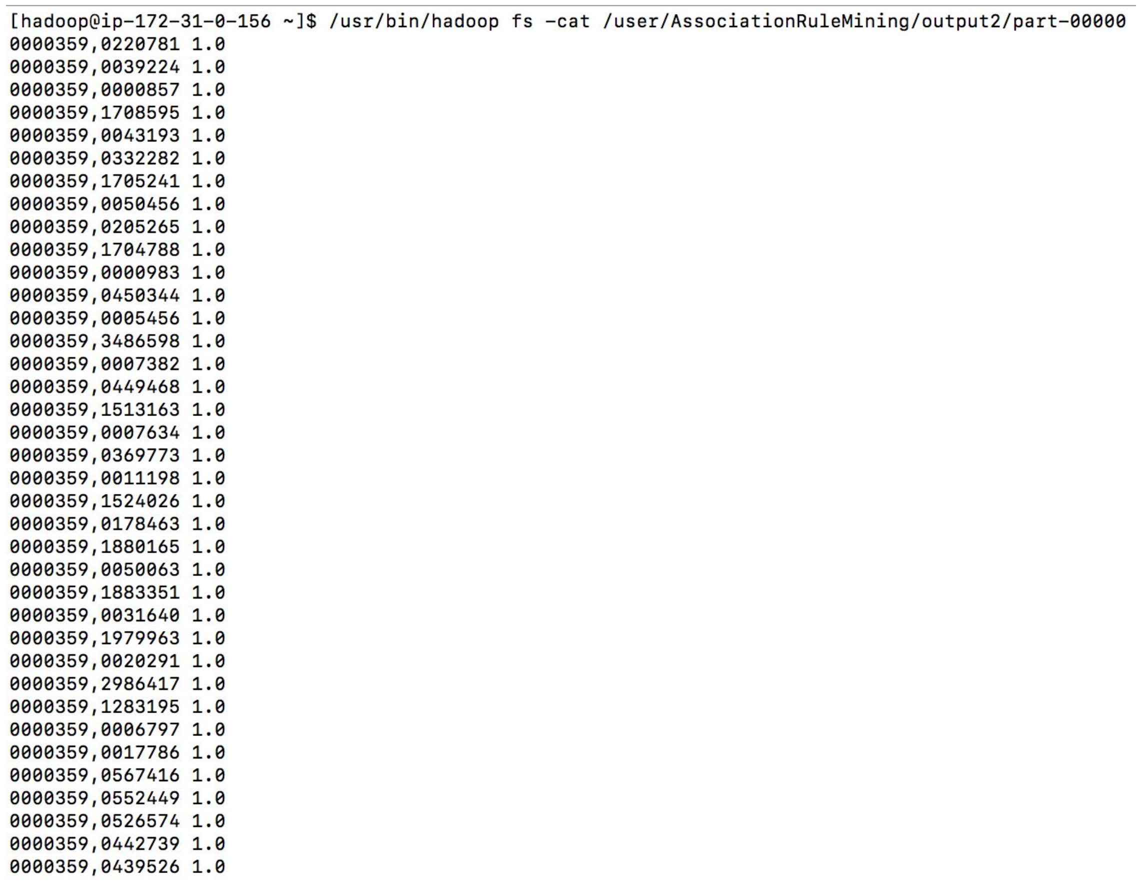
```
18/04/29 23:16:18 INFO mapreduce.Job:  map 100% reduce 100%
18/04/29 23:16:19 INFO mapreduce.Job: Job job_1525040612457_0008 completed successfully
18/04/29 23:16:19 INFO mapreduce.Job: Counters: 52
        File System Counters
                FILE: Number of bytes read=422582059
                FILE: Number of bytes written=868222940
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=1009240937
                HDFS: Number of bytes written=1802628824
                HDFS: Number of read operations=297
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=62
        Job Counters
                Killed map tasks=1
                Killed reduce tasks=1
                Launched map tasks=68
                Launched reduce tasks=32
                Data-local map tasks=63
                Rack-local map tasks=5
                Total time spent by all maps in occupied slots (ms)=81521952
                Total time spent by all reduces in occupied slots (ms)=37445088
                Total time spent by all map tasks (ms)=1698374
                Total time spent by all reduce tasks (ms)=390053
                Total vcore-milliseconds taken by all map tasks=1698374
                Total vcore-milliseconds taken by all reduce tasks=390053
                Total megabyte-milliseconds taken by all map tasks=2608702464
                Total megabyte-milliseconds taken by all reduce tasks=1198242816
        Map-Reduce Framework
                Map input records=55683241
                Map output records=55683241
                Map output bytes=1007089304
                Map output materialized bytes=428927887
                Input split bytes=10404
                Combine input records=0
                Combine output records=0
                Reduce input groups=42343
                Reduce shuffle bytes=428927887
                Reduce input records=55683241
                Reduce output records=55640898
                Spilled Records=111366482
                Shuffled Maps =2108
                Failed Shuffles=0
                Merged Map outputs=2108
                GC time elapsed (ms)=33714
```

The Total Execution time



```
Job Step Two took 76721 milliseconds
Both Jobs took 926792 milliseconds
[hadoop@ip-172-31-9-231 AssociationRuleMining]$
```

Sample Output



```
[hadoop@ip-172-31-0-156 ~]$ /usr/bin/hadoop fs -cat /user/AssociationRuleMining/output2/part-00000
0000359,0220781 1.0
0000359,0039224 1.0
0000359,0000857 1.0
0000359,1708595 1.0
0000359,0043193 1.0
0000359,0332282 1.0
0000359,1705241 1.0
0000359,0050456 1.0
0000359,0205265 1.0
0000359,1704788 1.0
0000359,0000983 1.0
0000359,0450344 1.0
0000359,0005456 1.0
0000359,3486598 1.0
0000359,0007382 1.0
0000359,0449468 1.0
0000359,1513163 1.0
0000359,0007634 1.0
0000359,0369773 1.0
0000359,0011198 1.0
0000359,1524026 1.0
0000359,0178463 1.0
0000359,1880165 1.0
0000359,0050063 1.0
0000359,1883351 1.0
0000359,0031640 1.0
0000359,1979963 1.0
0000359,0020291 1.0
0000359,2986417 1.0
0000359,1283195 1.0
0000359,0006797 1.0
0000359,0017786 1.0
0000359,0567416 1.0
0000359,0552449 1.0
0000359,0526574 1.0
0000359,0442739 1.0
0000359,0439526 1.0
```