New York Taxi Data Analysis

Zhewen Shi

May 29, 2015

Data can be downloaded from http://www.andresmh.com/nyctaxitrips/ and there is a description of the fields at http://publish.illinois.edu/dbwork/open-data/.

1. Deciles of the total amount less the tolls

Since the data is very large, all methods I used is parallelized. In this part, I used three methods to calculate the deciles: shell + r; c + r; hadoop + r. All methods get the same results as follows:

```
##   0.1   0.2   0.3   0.4   0.5   0.6   0.7   0.8   0.9
##  6.00  7.50  8.50  9.75 11.00 13.00 15.00 18.50 26.12
```

1.1 Methods

1.1.1 Method1: shell + r

The framework of method in R is as follows:

Algorithm 1 Framework of method in R

1: $f \leftarrow$ list of all trip_fare files
2: $cl \leftarrow$ makeCluster(12, "FORK")
3: $els \leftarrow$ clusterSplit(cl, f), assign each file to one node
4: $wc \leftarrow$ clusterApply(cl, els, function(x) getFee(x)), in each node, run getFee function to get total amount less toll amount frequency table
5: merge wc together to get one frequency table
6: get quantiles from frequency table

The pseudo-code of getFee function is as follows:

Algorithm 2 getFee(filename)

1: $fee \leftarrow$ the column of total amount less tolls amount
2: $fee \leftarrow$ round fee with two decimal places
3: return(table(fee))

Shell is used to calculate fee(the column of total amount less tolls amount in each trip_fare file).

1.1.2 Method2: c + r

The pseudo-code of getFee function is as follows:

Algorithm 3 getFee(filename)

1: $numLines \leftarrow$ the number of lines in the file
2: $fee \leftarrow$ a 0 list with the length numLines - 1
3: $output \leftarrow$ .C("getFee", as.character(filename),as.numeric(fee))
4: $fee \leftarrow$ round output[[2]] with two decimal places
5: return(table(fee))

This method is similar with 1.1.1. The difference is that I used c code to calculate fee(the column of total amount less tolls amount in each trip_fare file).

1.1.3 Method3: hadoop + r

In this method, I used hadoop to calculate total amount less toll amount frequency table of all files. It is a basic application of hadoop. During mapper, we parse each line and check if it is the header of each file. If not, calculate total amount less tolls in that line and set the result as key and value is 1. Then, during reducer, compute the key frequency. After all done, use getmerge to download output files and use R to get quantiles.

1.2 Time Comparison

System.time	user	system	elapsed	Sever
c + r	270.152	13.388	26.760	poisson
shell + r	1578.376	76.228	223.160	poisson
hadoop			840	hadoop

Table 1: system.time(in seconds)

We can see that c + r is the fastest method. The reason is that c is much faster than script language shell when reading large data file. In our department, the hadoop sever is too old. It is much weaker than poisson sever. So, althouth the running time is longest whn using hadoop, we can not conclude that the hadoop method is weaker than c + r.

Then, let us check the runing time of parallelized program and serialized program. Parallelized program means using makeCluster in R to make a cluster which contains 12 nodes. Each node read one file and get the total amount less tolls. Serialized program means using one master CPU to run files one by one.

The parallelized code is as follows:(f is a list which contains all trip_fare file names)

```
cl = makeCluster(12, "FORK")
els =  clusterSplit(cl, f)
system.time(clusterApply(cl, els, function(x) getFee(x)))
```

The serialized code is as follows:(f is a list which contains all trip_fare file names)

```
system.time(lapply(1:12, function(i) {getFee(f[i])}))
```

The running time is as follows:

System.time(in seconds)	user	system	elapsed	Sever
c + r (parallelized)	0.021	0.006	52.026	poisson
c + r (serialized)	284.037	21.747	419.554	poisson

The user time of parallelized code is quite small. Although the documents about system.time said that "user time is the total user times of the current R process and any child processes on which it has waited", I think such a small user time only means the time master node used. Acutally, master node only assign tasks to slave nodes and wait for their results. During the waiting time, the process on master node has been blocked and does not use CPU. I did not know why user time of each child process in savle node had not been returned, may be the performance of server is not stable.

Since two methods were run in the similar period of time, I suppose the environment is the same. If we use elapsed time, we can see that the running time of serialized code is less than 12 times of running time of parallelized code. The reason is that it need time to communicate between salves and master. All 12 salves want to read file from hard disc. Sometime they may wait because of the I/O limit.

1.3 Summary

I used three hours to finish the first and second method. When using hadoop, I used four hours. The challenge when finishing the first two methods is that how to merge 12 frequency table together quickly. I used data.frame as an intermediate value which contains two columns, one is the different values of total amount less tolls, the other is their frequency. Then, the aggregate function in plyr package can be used to get the merged frequency data.frame.

It is happy that when using hadoop, there is no need to merge 12 frequency tables. Hadoop can put the same key in the same reducer node. However, I used much time to do mapper, as follows:

15/05/22 23:53:29 INFO streaming.StreamJob: map 0% reduce 0%

15/05/22 23:53:39 INFO streaming.StreamJob: map 2% reduce 0%

15/05/23 00:04:54 INFO streaming.StreamJob: map 100% reduce 13%

15/05/23 00:07:28 INFO streaming.StreamJob: map 100% reduce 100%

15/05/23 00:07:29 INFO streaming.StreamJob: Job complete:

But the mapper funciton is just read each line and do total amount less tolls. May be I need to use a better hadoop sever.

Among these three methods, shell is easy to use, but the running time is longer. C is a little harder to use because of the pointer, but it is much faster. I did not use any serialized method because the data is large.

2. Linear regression

2.1 Check whether each row of trip data file and trip fare file matched

I used four columns to check if they matched line by line. They are medallion, hack_license, vendor_id and pickup time. Similar with problem 1, I used R to make a cluster with 12 nodes. Then, in each node, open one pair of trip_fare file and trip_data file to check whether they matched. The check function runing on each node is written by c. The pseudo-code of c function is as follows:

Algorithm 4 check(char **filename_fare, char **filename_data, int *unequal)

1: $*file\_fare \leftarrow$ fopen(*filename_fare, "r")
2: $*file\_data \leftarrow$ fopen(*filename_data, "r")
3: get one line by file_fare pointer and copy the content to buffer1;
4: get one line by file_data pointer and copy the content to buffer;
5: $nl \leftarrow 0$
6: **while** file_fare and file_data do not reach the end of files **do**
7: get one line by file_fare pointer and copy the content to buffer1;
8: get one line by file_data pointer and copy the content to buffer;
9: Using strsep to split the content in buffer1 and buffer2 and get medallion, hack_license, vendor_id and pickup time separately
10: **if** all medallion, hack_license, vendor_id and pickup time **then**
11: $unequal[nl] \leftarrow 0$
12: **else**
13: $unequal[nl] \leftarrow 1$
14: **end if**
15: $nl \leftarrow$ nl + 1
16: **end while**
17: close file_fare and file_data

The running time is as follows:

System.time(in seconds)	user	system	elapsed	Sever
	0.045	0.045	58.828	poisson

Since it is fast (If there is no other users, the elapsed time is around 10 seconds), I did not use sample method. The challange is the c code. First, I used strtok function to split each line. However, this function can not deal with missing values. If there is a missing value in the file, there are two successive commas. strtok function filter two commas at one time. Then the value I get belongs to the next column. The code can run correctly after I used strsep function instead of strtok.

2.2 Linear function

2.2.1 Method 1: c + r

This method is also similar with prob1. The difference is that we use clusterApply function twice in R.

At first time, let each slave node read one pair of files, get trip time(X) and total amount less tolls(Y), calculate sum(X), sum(Y), sum(X^2) and sum(XY). However, I did not let slave node return all data to master node. Instead, keep X and Y in each slave node as global variables and return sum(X), sum(Y), sum(X^2), sum(XY) and n(number of lines) to master process. Then in the master process, calculate beta_0, beta_1 by the following formula.

$\hat{\beta}_1 = \frac{\Sigma X_i Y_i - n\bar{X}\bar{Y}}{\Sigma X_i^2 - n*\bar{X}^2}$, $\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1\bar{X}$

After that, send beta_1, beta_0 and ymean to 12 slave nodes and calculate residuals and anova table.

The results is as follows: $\beta_0 = 14.52334, \beta_1 = 2.06023 * 10^{-5}$.

	SS	d.f.	MS	F*
Regression	SSR=1.101323e+14	d.f.(SSR) = 1	MSR=1.101323e+14	F* = 23199650585
Error	SSE=822111200476	d.f.(SSE) = 173179757	MSE=4747.155	
Total	SSTO=1.109545e+14	d.f.(SSTO) = 173179758		

Table 2: ANOVA table of linear regression

2.2.2 Method: biglm

In this method, I did not calculate beta_0 and beta_1 by formula. Instead, I used biglm function in biglm package to get the result. We can use update function to update the linear model by adding new data in it.

The result is the same with 2.2.1.

	Coef	(0.95	CI)	SE
(Intercept)	1.452334e+01	1.451285e+01	1.453382e+01	5.242517e-03
trip_time	2.060230e-05	1.994021e-05	2.126439e-05	3.310456e-07

Table 3: Linear regression result of biglm

2.2.3 Comparision and analysis

The running time of these two methods are as follows:

System.time	user	system	elapsed	Sever
c + r(Method1)	0.025	0.017	15.040	hilbert
biglm(Mehtod2)	398.211	36.376	294.526	hilbert

Table 4: system.time(in seconds)

In method1, the running times does not contain ANOVA table calculation. Although the running time of method2 is much longer than method1, it contains many other informations such as confidence interval. And biglm method is much faster than lm. Another reason why method 2 is slower than method 1 is that when using clusterApply function, method 1 only need to return sum informations to master node, but method 2 needs to return all trip time and fee to master node.

There is no challange in this part.I used two or three hours to finish them. However, since sometimes there are too many people using sever at the same time, sometime the codes runs slow(elapsed time).

3. Multiple regression

Similar with problem 2, I used two methods. In the first method, I calculated coefficients by formula. Then, in the second method, I used biglm to get the result. The programming time here is less than one hour.

The result of method 2 is as follows. The method 1 get the same coefficients.

	Coef	(0.95	CI)	SE
(Intercept)	1.446943e+01	1.445613e+01	1.448272e+01	6.646165e-03
trip_time	2.061321e-05	1.995112e-05	2.127530e-05	3.310465e-07
surcharge	1.687918e-01	1.432123e-01	1.943713e-01	1.278974e-02

Table 5: Multiple Linear regression result of biglm

The running time is as follows:

System.time	user	system	elapsed	Sever
c + r(Method1)	0.030	0.021	15.706	hilbert
biglm(Mehtod2)	391.909	36.578	285.519	hilbert

Table 6: system.time(in seconds)

As described in problem 2, method 1 is much faster than method 2. However, method 2 get more information than method 1, such as SE and confidence intervals. In addition, method 2 is more flexible than method 1. If we want to add new regressors, we do not need to change much code in method 2. However, if we use method 1, we need to know the formula.