Part 2: Cardinality Estimation program

921808
Qianyu Guo

section 1.Introduction

After running the code provided by teaching stuff, which includes multiple copies of the AMS estimator, the results show that the when the number of created hash function is small, the result is still not accurate, since in the algorithm, we use median to predict the distinct number, the result become more accurate if we use more hash tables. However, more hash function means more time and space. This report aims to find a balanced point between accuracy, space, and running time by changing the number of hash functions.

section 2. The test data sets
1.file size:78kb (including integers repeated twenty times from 0 to 999)
2.file size:150MB(including integers repeated many times from 0 to 9999)
3.file size:150MB(including random numbers in range 0 to 9999)

section 3. Methods for measuring running time, space, and accuracy
1). Methods for measuring running time
By using java code:

```
long startTime=System.currentTimeMillis();//get start time
(program running)
long endTime=System.currentTimeMillis(); //get end time
System.out.println("running time：  "+(endTime-startTime)+"ms");
```

2). Methods for measuring space
By using java code to record the space consumed by each call to new:

```
System.gc();
Long beforeUsedMem
=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
(Distinct ams = new AMS();)
long afterUsedMem
=Runtime.getRuntime().totalMemory()-Runtime.getRuntime().freeMemory();
long newMemo =afterUsedMem-beforeUsedMem;
System.out.print("new consume"+newMemo);
```

3). Methods for measuring accuracy
At most of the time the program returns the same result that not far away from the actual distinct number, when a outlier occurs, record it and calculate the fraction of outlier

in all the result, and calculate standard deviation.

section 4. Experimental procedure

First use the small size evenly distributed file as input, by changing the different parameter c in JAVA code: "in k = (int) Math.ceil(c*Math.log(del1)); ", to see how many hash functions are enough to get the high accuracy of the algorithm. And set this as the lower bound for a much larger file to get accuracy result.

Then use the parameter to run the file with a larger range of evenly distributed numbers , continue to adjust parameter c according to accuracy, then draw the diagram of the result.

Use the result above to run the file contain random numbers, and adjust parameter to find the balance of accuracy, space, and running time.

Finally compare the algorithm performance with the initial parameter.

section 5 Results and analysis

1) Accuracy result of small size evenly distributed file:

file size:78kb (including integers repeated twenty times from 0 to 999)

c	18	30	35	45	50	60
k	54	90	105	120	150	180
Fraction	76%	83%	89%	92%	91%	96%
Standard Deviation	709	597	480	409	434	289

Table 1: result of dataset 1

In the above table, the fraction means the fraction apart from outlier in all the result. We notice that starting from 40, the fraction apart from outlier is more than 90%, so we can use it as the lower bound of our next experiment.

2) Accuracy versus time and space of big size evenly distributed file:

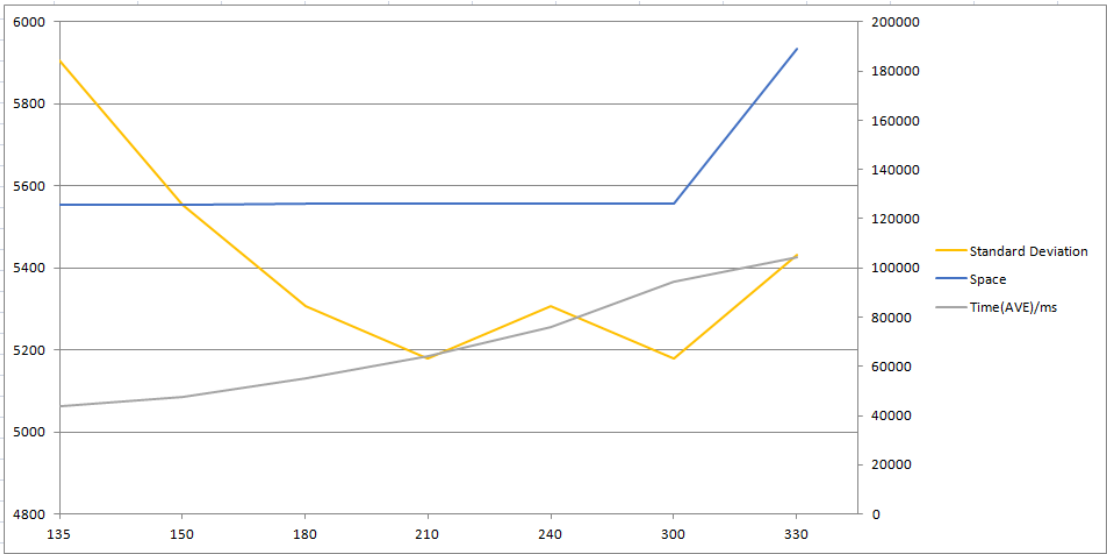
file size:150MB(including integers repeated many times from 0 to 9999)

C	45	50	60	70	80	100	110
K	135	150	180	210	240	300	330
Fraction	74%	77%	79%	80%	79%	80%	78%
Standard Deviation	5907	5555	5308	5180	5308	5180	5433
Space	125800	125568	125936	125936	126048	125936	188896
Time(AVE)/ms	43744	47578	54993	64301	75921	94486	104476

Table 2: result of dataset 2

As we can see above, compared with small size file, when c and k stay the same, the accuracy of big size file with bigger range has decreased. And starting from 60, the

accuracy stays almost same, and when c increase by 10 the running time is about to add 1000ms.



Plot1: the relation between Time standard deviation and k

As we can see above, memory consumption has remained almost the same, except for the last point. It may because some objects might reasonably have been garbage collected. During the experiment, the parameter of jvm is: -Xmx3550m -Xms3550m -Xmn2g -XX:SurvivorRatio=8 . So here we cannot find any relationship between memory and accuracy, or maybe because the parameters we set are small and experimental error exists.

If we don't consider the weight of each factors, we may can conclude that a good point to balance accuracy and time is in range of c = 60(where k=180) and c =70(where k=210). Still, there might be some k values for bigger parameter c that can make a good balance among accuracy, space and time.

3) Accuracy, time and space of big size random file:

file size:150MB (including random numbers in range 0 to 9999)

C	60	70
K	180	210
Fraction	77%	75%
Standard Deviation	5555	5792
Space	126016	126048
Time(AVE)/ms	52286	64850

Table 3: result of dataset 3

As we can see above, when we use dataset 3, we can see that the algorithm performs almost the same when c=60(where k=180) and c =70(where k=210). But the result of the former is slightly better than the latter. More importantly, the former spends less time. So we consider c=60(where k=180) as our balanced point among time, space and accuracy.

4)Compare with initial parameter c =18:

C	18	60
K	54	180
Fraction	65%	77%
Standard Deviation	6853	5555
Space	125800	126016
Time(AVE)/ms	17151	52286

Table 4: result of initial parameter

Compare with c=18(where k=54), using c=60 needs more time and space, but have higher accuracy.

section 6 Conclusion

The results show that to some extent, when the number of hash functions becomes larger, the result can be more accurate.

Also, the file size and item range will affect the accuracy of the algorithm, in this experiment, we use range 0 to 99999 as distinct number, and file size we use is about 150MB. Under this circumstance, when c = 60(where k=180) is a good balanced point among time, space and accuracy. However, if the range becomes larger, we need more hash tables.