Distinct-Values Estimation in Large Datasets

Tihomir Elek

December 2015

Abstract

Estimating cardinality of large datasets is a cornerstone of big data research and a useful tool in query optimization methods for relational databases. Counting number of unique elements in a large volume of data is an expensive task. Abundant research of cardinality estimation exists and is divided into stream based and sampling based algorithms. Stream based algorithms scan the whole input once and estimate the result. Sampling based algorithms scan a small fraction of the data and determine the cardinality of the whole set using statistical methods. The main trade-off in these algorithms is between memory use and the accuracy of the estimation. I have tested and benchmarked some of the most notable stream based algorithms in use, including Flajolet’s Adaptive Sampling and HyperLogLog algorithms. A remarkable feature of these algorithms is that they are designed in such a way that the memory usage or precision are not affected by the actual value of the cardinality. I have also implemented and compared the accuracy of Adaptive Sampling with Self-Learning Bitmap to other algorithms. Adaptive Sampling with Self-Learning Bitmap uses an interesting approach to sampling which is very accurate, but only for relatively small cardinalities. I have tested the algorithms on cardinalities ranging from 1 to \(10^{10}\) and reported the mean standard error for each algorithm out of 100-500 trials for each cardinality. I also researched sampling based algorithms and found a theorem with provable inaccuracy of sampling based algorithms for large distribution of inputs, which could be the subject of my future work.

1 Introduction

Advancements in storage technologies led to decreased costs of storing vast amounts of data. Analyzing large amounts of data is an expensive computational task. Many datasets are too large to fit in the memory of a single machine and their analysis requires frequent access to the disk. Sending data over the network results in high latency. Hence, determining the number of distinct elements in the dataset becomes very expensive. Such problems resulted in the development of algorithms that utilize surprisingly small data structures to estimate the number of distinct elements in a set. Furthermore, almost all query optimization methods in relational databases require a fast way of determining
the number of distinct elements of some attribute. Based on the knowledge of
the number of distinct elements, it is possible to select an optimal query plan.
Doing a selection operation before join or vice-versa can change the query exe-
cution time dramatically.

To better illustrate the challenge of determining the number of distinct ele-
ments in a large set let’s consider a common task of analyzing logs. Suppose
you have a 16 or 32 character unique user ID (UUID) and you would like to
know how many distinct users interacted with your application. UUID will, de-
pending on the size, be represented by 128 or 256 bits. In case of a 32 charac-
ter UUID, 30 000 UUIDs require 1MB of space. 30 million requires 1GB. But, often
applications serve billions of events per day. In that case UUIDs will take up
hundreds of gigabytes of storage. The simplest way to count distinct UUIDs
would be to create a hash table and return all unique values from the table. But
machines don’t have hundreds of gigabytes of memory and even if they do, it is
shared amongst many processes. Imagine Google tried to determine number of
queries run in the past 2 years using hash tables. A virtually impossible task.
Another option is using bitmaps.[8]

Underlying idea of all algorithms using bitmaps is that the input is hashed
into a bit field. Each element is mapped to a bit in the bitmap. This approach
significantly improves the memory requirements for estimating cardinality but
bitmaps are often sparse and take up too much space. In the past three decades,
there has been extensive research on the problems of cardinality estimation.

2 Algorithms

There are two main directions in cardinality estimation research. Many algo-
rithms have been developed to use one full scan of the input data and derive
estimates of the number of distinct elements using only a small amount of mem-
ory and computational resources.

The other class of algorithms is sampling based algorithms that do not scan
the whole dataset, but rather a small fraction of data and use statistical analy-
sis to estimate the cardinality. Needless to say, these algorithms are less precise
than the full scan (stream based) algorithms, but require much less memory and
are executed much faster.

Most of the stream based algorithms use the following approaches or some com-
bination of the two: Sampling or Wegman’s adaptive sampling to get random
portions of values, assign each portion to a field in a bit vector and count the
empty fields of a bit vector in order to estimate the cardinality; Order statistics,
developed by hashing each value to a random number with a common distri-
bution.
The most notable stream based algorithms are Probabilistic Counting, LogLog, HyperLogLog(++), S-bitmap, Adaptive Sampling and Adaptive Sampling with Self-Learning Bitmap.

I will report on the performances of Adaptive Sampling, HyperLogLog and Adaptive Sampling with Self-Learning Bitmap (S-bitmap) in terms of memory and accuracy and benchmark them against each other. In the report, I describe my findings about the relationship between the characteristics of the input data and the performance of the algorithms. This area is an important part of database and data mining research, so the algorithms are well studied and their performances are well tested. However, Adaptive Sampling with Self-Learning Bitmap is a relatively new algorithm that might have the potential to be a better option for some use cases over HyperLogLog. So, I dedicated more time understanding and implementing it.

Secondly, I researched the sampling based algorithms that are not widely used and are less reliable for computing. I investigated how such algorithms could be used to estimate cardinality and how different datasets determine the output. After some research I found a paper which establishes that one cannot guarantee error bounds on sampling algorithms for all possible datasets. After that realization, I decided to focus more on stream based algorithms.

3 Adaptive Sampling

Adaptive Sampling is an algorithm proposed in 1985 by Philippe Flajolet in his paper “On Adaptive Sampling”[1]. The algorithm utilizes Wegman’s Adaptive Sampling method. It observes bits of hashed values of scanned records to estimate the number of distinct elements in the dataset. The result is probabilistic due to the choice of the hash function and it’s way of hashing the input data and several other parameters. The algorithm keeps a list of \( m \) hashed values. After the list of \( m \) values has been populated with hashed input values, the list of samples overflows and the “depth” is increased by 1. The list of \( m \) values is scanned and only certain elements are kept. Process is repeated until all input records are exhausted. The estimate of number of distinct elements is then:

\[
2^{depth} \times l
\]

where \( l \) is the cardinality of the list.

It is often the case that the data being analyzed is coming from clickstream or the internet of things and that the replication factors in the data are very large. On the other hand, some datasets only have unique elements in them. I investigated potential relationship of replication factor and the precision of the algorithm. After carefully examining the analysis of the proposed algorithm in Flajolet’s paper ”On Adaptive Sampling”, I realised that by construction, Adaptive Sampling is independent of the structure of replications of the input.
data. Algorithm keeps track of hashed values of inputs and in case of duplicate elements, the hashed value will already be in the list and won’t be entered again. That way the cardinality of the list won’t change, ”depth” won’t increase and the results will be within the accuracy predictions even in extreme cases in which a single element is repeated \( n \) times, or all elements are unique. In almost all papers, authors calculate the accuracy of cardinality estimation as the quotient of the square root of variance of the estimate and the exact value: \( V n^{1/2}/n \). \( V_n \) is according to Theorem 1 proved in [1]:

\[
\frac{V_n}{n^2} = \frac{1}{(m - 1)\log 2} + P(\log_2 n) + o(1)
\]

where \( P(u) = \sum_{k \in \{0\}} p_k e^{-2iku} \), such that

\[
p_k = \frac{1}{\log 2} \Gamma(-1 + \frac{2ik\pi}{\log 2}) \left(\frac{2ik\pi/\log 2 + m - 2}{m - 1}\right)
\]

We can see that the quotient of the standard deviation by exact cardinality, which is also referred to as standard error is a value that depends on \( m \), with very little dependence on actual cardinality \( n \). This feature of the algorithm is very important, since it allows for usage of a small amount of memory for very large data sets. According to the theorem, larger \( m \) will yield more accurate results. The analysis of the algorithm in the the original paper establishes that the Adaptive Sampling is unbiased and that its relative accuracy is expected to be around \( 1.20/\sqrt{m} \).

I used an existing implementation of Adaptive Sampling to test the algorithm. The measure I used for accuracy benchmarking is percentage standard error, defined above as \( V n^{1/2}/n \). The results can be found in Figure 1. All tests of given cardinality are executed on files that were randomly taken as samples from larger randomly generated files and repeated 100-500 times for each given cardinality and \( m \) value. Mean results are presented in the tables. Algorithm performances were also tested on simulated on simulated real datasets created at the Interana.

<table>
<thead>
<tr>
<th>( N_{max} )</th>
<th>( m = 2^{14} )</th>
<th>( m = 2^{15} )</th>
<th>( m = 2^{16} )</th>
<th>( m = 2^{17} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 10^4 )</td>
<td>5.3</td>
<td>3.76</td>
<td>2.65</td>
<td>1.87</td>
</tr>
<tr>
<td>( 10^5 )</td>
<td>5.34</td>
<td>3.7</td>
<td>2.63</td>
<td>1.8</td>
</tr>
<tr>
<td>( 10^6 )</td>
<td>5.76</td>
<td>3.9</td>
<td>2.65</td>
<td>1.86</td>
</tr>
<tr>
<td>( 10^7 )</td>
<td>5.31</td>
<td>4.04</td>
<td>2.71</td>
<td>1.91</td>
</tr>
<tr>
<td>( 10^8 )</td>
<td>5.95</td>
<td>3.96</td>
<td>2.78</td>
<td>1.94</td>
</tr>
<tr>
<td>( 10^9 )</td>
<td>5.69</td>
<td>4.3</td>
<td>2.84</td>
<td>1.96</td>
</tr>
</tbody>
</table>

Figure 1: Standard error (%) in Adaptive Sampling

---

1Tihomir Elek was a Software Engineering Intern at Interana in Summer 2015.
As expected, the results validated Theorem 1 and the estimates of standard error from the original paper. Almost all mean results agreed with the $1.20/\sqrt{m}$ precision. Adaptive Sampling executed very fast for all tested datasets. On the simulated real datasets, Adaptive Sampling proved to be very useful in counting unique elements, since it was accurate for a wide range of values and had no detectable bias for small values of $n$, which is often problem with algorithms from HyperLogLog family.

4 HyperLogLog

HyperLogLog algorithm is published by Flajolet’s group in 2007[2] and similarly to Adaptive Sampling, HyperLogLog relies on randomization which is ensured using hash functions and either bit-pattern observables or order statistics observables to estimate cardinality. As the name itself explains, this algorithm uses

$$\log\log(N_{max}) + O(1)$$

bits to estimate the cardinality $N_{max}$. Like the Adaptive Sampling, HyperLogLog allows changing of desired accuracy through $m$. HyperLogLog takes a data stream as an input, applies hash function to it, splits hashed input into $m$ substrings and maintains $m$ observables for each of the substrings.

Using bit-pattern observable and stochastic averaging, it averages the observables to obtain the cardinality whose accuracy improves as $m$ grows. HyperLogLog paper [2] also measures accuracy using standard error and presents a similar theorem to Theorem 1 in Adaptive Sampling. Accuracy is almost independent of $n$ and increases with $m$. Standard error of the HyperLogLog algorithm is estimated to be close to $1.04/\sqrt{m}$.

For the testing of HyperLogLog algorithm I modified an implementation found online.[3]. Results are presented in Figure 2.

<table>
<thead>
<tr>
<th>$N_{max}$</th>
<th>$m = 2^{14}$</th>
<th>$m = 2^{15}$</th>
<th>$m = 2^{16}$</th>
<th>$m = 2^{17}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^4$</td>
<td>2.17</td>
<td>2.1</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1.81</td>
<td>1.3</td>
<td>1.1</td>
<td>0.6</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1.82</td>
<td>1.3</td>
<td>1.3</td>
<td>0.64</td>
</tr>
<tr>
<td>$10^7$</td>
<td>1.82</td>
<td>1.76</td>
<td>1.3</td>
<td>0.6</td>
</tr>
<tr>
<td>$10^8$</td>
<td>2</td>
<td>1.64</td>
<td>1.1</td>
<td>0.7</td>
</tr>
<tr>
<td>$10^9$</td>
<td>2</td>
<td>1.74</td>
<td>1.6</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Figure 2: Standard error (%) in HyperLogLog algorithm

Results in Figure 2 show that the memory required to accurately determine the cardinality of the set of up to $10^9$ elements is less than 16 kilobytes.
An important addition to this algorithm are range corrections. Older version of the algorithm (LogLog) was inaccurate on small cardinalities and several improvements are introduced in HyperLogLog. According to the paper, HyperLogLog demonstrates no bias for cardinalities greater than $2^m$. However, for cardinality values of $n < \frac{1}{2}m$, nonlinear distortions appear, resulting in inaccurate cardinalities. That could be seen for the $10^4$ cardinality in the Figure 2 where the mean standard error did not completely follow the prediction.

Without the range corrections, my implementation in extreme cases, produced a maximum standard error close to 1500% and average standard error of 100%. Some of these issues are solved and the potential for better scaling is added by Google’s research team with HyperLogLog++[4].

5 Adaptive Sampling With Self-Learning Bitmap

As I was investigating HyperLogLog and other methods used in cardinality estimation, I found a somewhat unique approach to Adaptive Sampling. It is proposed in the paper written by Aiyou Chen and Jin Cao, “Distinct Counting with a Self-Learning Bitmap”[5]. Authors claim that their algorithm is uniformly reliable, i.e. invariant to unknown cardinalities and given the same memory requirement as HyperLogLog and Adaptive Sampling and that it performs better on a range of inputs.

The uniqueness of this adaptive sampling algorithm comes from the sampling idea. Differently from Wegman’s Adaptive Sampling used in the Adaptive Sampling algorithm, S-bitmap learns the sampling rates from the number of distinct elements already encountered and adjusts the estimation accordingly. I found that idea very interesting. Data analysis requires constant accuracy. Given Adaptive Sampling was occasionally more accurate than HyperLogLog, I investigated if S-bitmap could potentially be used as a more accurate version of adaptive sampling.

I implemented the algorithm proposed in [5] and analyzed its performance in a similar fashion as the previous two. I confirmed the correctness of the algorithm by running the test cases on a similar implementation found on Github.[7] In case of S-bitmap the values for $m$ were chosen by the algorithm according to the relation from [5]

$$m = \frac{\log(1 + 2N_{\text{max}}\epsilon^2)}{\log(1 + 2\epsilon^2(1 - \epsilon^2)^{-1})}$$

where $N_{\text{max}}$ is maximum possible cardinality and $\epsilon$ desired estimation precision.

The difference in obtaining results between the HLL, AS and S-bitmap is that in HLL and AS I varied $N_{\text{max}}$ and $m$ to obtain accuracy and in S-bitmap, for comparison purposes, I varied desired estimation accuracy and $N_{\text{max}}$ to obtain $m$ used by the algorithm. The results are presented in Figure 3.
My experimental results confirm the results from the original paper. Comparing the data in Figures 1 and 2 to Figure 3, we can conclude that S-bitmap does outperform both Adaptive Sampling and HyperLogLog for cardinalities up to $10^8$, However, the memory required to maintain precision is growing more rapidly than for the other two algorithms which makes this algorithm less likely to scale for cardinalities larger than $10^9$. S-bitmap outperforms Adaptive Sampling for the test cases presented, but for large cardinalities of more than $10^{12}$, it is unlikely to be practical.

### 6 Sampling Based Algorithms

I wanted to find a sampling based algorithm that would be precise enough to be used for data analysis and whose accuracy would be consistent with high probability for all types of input. However, in a paper by Charikar and Chaudhuri ”Towards Estimation Error Guarantees for Distinct Values”[6], they prove that no estimator can guarantee small error across all input distributions without scanning a large fraction of the input dataset.

After understanding their theorem I decided to focus more on stream based estimators. However, I might continue my inquiries on this topic in my future work.

### 7 Conclusions and Future Work

Adaptive Sampling and HyperLogLog are being used by many systems and applications and each have their own advantages and disadvantages. HyperLogLog uses less memory for the same accuracy and scales well, but is somewhat inaccurate on small values of $n$. Adaptive Sampling is simpler to implement, executes faster and performs constantly well, but requires more memory than HyperLogLog to obtain the same precision. S-bitmap performs really well for relatively small cardinalities, but doesn’t scale well.

<table>
<thead>
<tr>
<th>$N_{max}$</th>
<th>$m = 2^{14}$</th>
<th>$m = 2^{15}$</th>
<th>$m = 2^{16}$</th>
<th>$m = 2^{17}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^4$</td>
<td>0.81</td>
<td>0.56</td>
<td>0.21</td>
<td>0.1</td>
</tr>
<tr>
<td>$10^5$</td>
<td>0.88</td>
<td>0.64</td>
<td>0.3</td>
<td>0.14</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1.3</td>
<td>0.88</td>
<td>0.45</td>
<td>0.25</td>
</tr>
<tr>
<td>$10^7$</td>
<td>1.3</td>
<td>1.1</td>
<td>0.6</td>
<td>0.31</td>
</tr>
<tr>
<td>$10^8$</td>
<td>1.79</td>
<td>1.32</td>
<td>0.89</td>
<td>0.6</td>
</tr>
<tr>
<td>$10^9$</td>
<td>2.3</td>
<td>1.50</td>
<td>1</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Figure 3: Standard error (%) in S-bitmap
Cardinality estimation can be very useful in analyzing large datasets and optimizing query execution in relational databases. Current stream based algorithms are accurate with high probability and their accuracy and memory usage keeps improving. This is necessary as the volume of data being stored is rapidly increasing.

Sample based algorithms seem to have reached a limit in this use case, but moving forward I would like to investigate the possibility of a hybrid approach to sampling algorithms, in that the algorithm used would be determined based on the distribution of the input data.

8 Acknowledgments

I would like to thank my project adviser Ruzica Piskac for her continued support and guidance and Viktor Barinov from Interana for providing advice about the algorithms and test datasets.

References


