Adding Sampling in Probabilistic Counting Algorithm

Rishabh Pandita
Department of Computer Science, University of Florida
rishabhpandita@ufl.edu

***Abstract*—This paper presents an approach for introducing a sampling module in Probabilistic Counting Algorithm (PCA). In a resource crunch environment when our priority is saving as much memory as we can, our results may show aberrations. In order to reduce the aberrations we have to make some compromise between performance and accuracy. It should be brought to the notice of readers that an pre-known error rate in results is always acceptable when it improves performance by reducing memory consumption while processing. This paper is aimed at achieving that balance, namely between the above two mentioned parameters by introducing a sampling module in PCA. Topics covered in paper include introducing to the problem and a solution for it and then deriving it mathematically, showcasing the experiment which was performed to compare the results of the new sampling module along the PCA algorithm and the original PCA without any sampling module. A table showing different sampling rates along with respective error rate was then made so as to represent which is the best sampling rate we can use.**

Keywords—PCA; Bitmap;Sampling; Error detection;Ttraffic measurtment counting flows; Measurment; Network Operations;Algorithims;

I. INTRODUCTION

A. Motivation

In modern world, information transfer happens fast and through devices that can fit into our pockets. To makes things small and fast we need less memory and high processing. These two would have been considered to be at two extreme of a tunnel a few decades back when computers used to take size of room but not now. Measuring internet traffic is one of the processor intensive tasks which is necessary to dive into the pool of information and find out what is important to users. PCA algorithm takes an innovative approach to this problem and uses probabilistic model to find the cardinality of data set. When our data set is real time and of a large size PCA does not perform well because of the limited memory. In order to achieve more accurate results, we need to be more considerate about the memory resource that we allocate to it which is not always possible. This urge to avoid expansion of memory for PCA and still maintain the accuracy needed gave rise to my motivation of introducing a sampling algorithm to it. Next section explains how PCA works.

B. General Measurment (Liner Counting)

This is basic line counting algorithm. First let us establish that we have to measure unique number of elements in a flow. To demonstrate this consider we have a large dataset of source and destination addresses. We have a unique source IP to which a destination IP maps so we choose source IP to be our flow ID. Now we need to find out how many unique destination IP maps to this flow ID (source IP). It is given that duplicate destination IP reside in a dataset of which we have a unique flow ID. To solve this we use the information provided by paper "Bitmap Algorithm for Counting Active Flows on High Speed Links" and section two of A Linear - Time Probabilistic Counting Algorithm for Database Algorithm. We take a large bitmap and use a randomized hash function to generate an index by taking destination ID as input and turning on that bit. This is called as online operation for PCA. It is

really important to use a good randomized hash function which can produce a large spread of numbers from 0 to the size of the bitmap. It is also important to decide the size of the bitmap to improve the efficiency of algorithm. After online operation follows offline operation which includes counting the number of zero in the bit map and applying formula 1 to get an estimation of size of the bitmap.

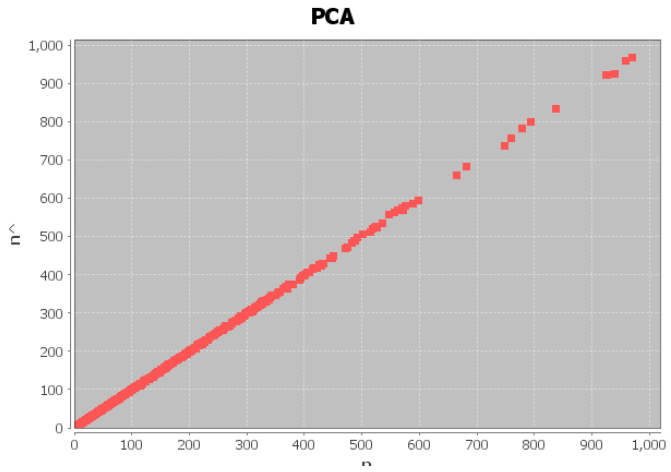
$$\hat{n} = -m \ln V_n. \quad (1)$$

In the above formula (1) m is the size of bitmap and Vn is the number of zeros in the bitmap after online operation divided by the size of bitmap (m). Thus we get an estimate of cardinality of unique elements in a set.

C. Limitations of using Liner Counting

Consider we have a large dataset and a small size of bitmap. When doing online operation all indices of bitmap will be set and thus our estimation will give bad results because our estimation is dependent on number of zero's in our bitmap. Figure (2) explains how having a small bitmap and large dataset will give wrong results. Having a small bitmap will result in hash collision after some time and resultant bitmap will have no zeros thus our Vn will be zero, since most of languages in which we implement PCA will give maximum value of the data type when evaluating log(0) our graph will result in constant line with estimated value as maximum limit of data type which holds it. Graph of figure (1) demonstrates the positive result of PCA when bitmap size is sufficiently large to accommodate the size of the dataset.

Figure 1



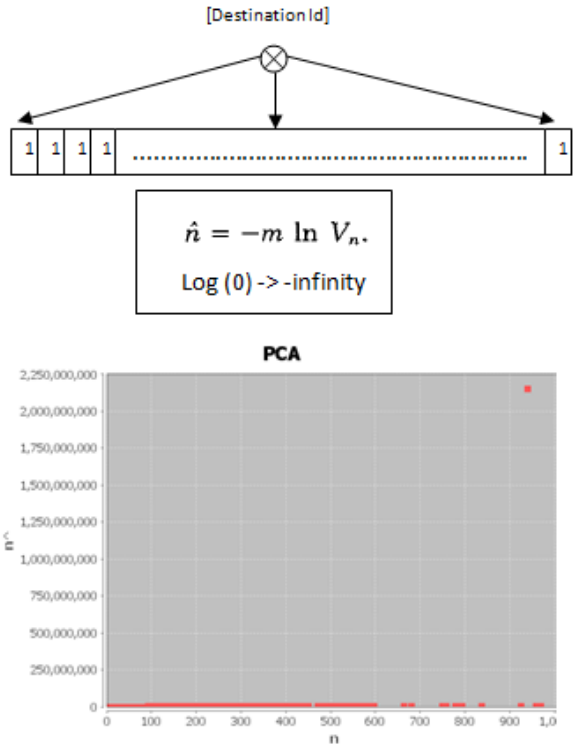


Figure 2

D. Techical Challanges

Now, an obvious question arises why do we need to reduce the number of size of bitmap? We can keep the size of bitmap sufficiently large so that it always respects the uniform distribution of elements and avoids hash collisions. To achieve this we need an excellent hashing function and spare memory. Both of the above mentioned things are expensive. When measuring traffic of internet we have billions of data to be measured thus it is a fact that in real life scenario our data set is huge. Because of huge data size our excellent hashing function will take time to produce a hash function result and it is against the design of our modern routers and gateway to carry huge onboard memory to measure the flow. Thus it is time consuming and economically unviable to have a large bitmap to measure traffic. Thus to overcome this technical challenge we need to optimise our algorithm to handle these cases.

E. Propsed Solution

In this section I will describe the hypothesis of the proposed solution without going into details as there is a dedicated section filled with details about that. What if we compromise on the accuracy by a pre-calculated percentage ? If we know by how much our estimated results are going to be off we can extrapolate their values and reconcile the

results. This is the basis for our solution. We choose a sampling rate and select packets at that rate to process. If our sampling rate is 50 percent we choose alternate packets to process, if our sampling rate is 30 percent we choose first three packets and then skip next 7 packets till the end of flow. In this way we reduce the data per flow. We are here employing systematic sampling and not random sampling because data is coming to us in real time and we are unaware how long the data set could be.

II. Related Work

The bases of this paper has been derived from work done by Flojet and Martin. We use PCA as a basis to compare the one with sampling module. Sampling formula which is introduced is based on the derivation done by the same paper. Some inspiration about implementing PCA error and variation is taken by the work of Cristian Estan, Mike Fisk paper. We have used systematic sampling, its reference has been taken from papers present online.

III. Sampling Implementation

To introduce a sampling module we need to select an appropriate sampling rate. There are a few ways to select it, one of which is to run it for different values on the bit size where original PCA fails to deliver.

A. Data structure

There are no changes in the data structure used, We still use a bitmap of M size and an highly efficient hashing algorithm which has a good spread. For our experimental purpose we have used SHA512.

B. Online Operation

This is an interesting part where sampling is utilised. As mentioned before we define sampling percentage as the percentage of data flow we agree to process. If we choose to select a high sampling rate then there will be not much difference as distribution and density of 1's in the bitmap would still be same. So we choose and experiment with moderate and low sampling rate. In common terms, we process (s) percent of data and skip (100-s) percent of dataflow and then again process s percent of data which is followed till the end of flow. Rest of the process of hashing and setting a bit to 1 in bitmap remains same as described in PCA algorithm. Below is the snippet (Figure 3) of the adjustments done to online operation in order to consider sampling rate into consideration.

```
// Processing for each flow
i = 0;
int count = 0;
int flowCount = 0;
while (i < currentFlowSize) {
    count++;
    if ((count % ((samplerate * 10) + 1)) == 0) {
        count = 0;
        i += (int) (10 - samplerate * 10);
    } else {
        flowCount++;
        probCounting.setBitArray(currentFlow.get(i));
    }
    i++;
}
```

Figure 3

C. Offline Operation

Once bits are set by using the newly introduced sampling algorithm we need to run our estimator. Let's say our sampling rate is p then our estimated cardinality value is derived from the formula (2). So how do we end up with this formula is shown in next section.

$$\hat{n} = \frac{-m \ln V_n}{p} \qquad (2)$$

Just to give a snippet of how the offline operation changed refer to figure (4)

```
nEstimate = (int) (((-1) * MaxBitArraySize * java.lang.Math.log(Vs))
            / (double) samplerate);
```

D. Derivation

This section utilises the derivation done by previous papers mentioned in references and related works and builds upon them to introduced the sampling module. So now we can derive $\bar{n}$ which is the estimated cardinality as below. Please notice how sample rate p has been introduced into the derivation.

Lets derive the estimated cardinality. Let Um be the count of zeros in bitmap and Aj is the probability that bit is zero till the end then 1Aj represents that event Aj happens.

$$Vm = (Um \div m) - 1$$

$$Prob\{Aj\} = \left(1 - \frac{1}{m}\right)^{np}$$

$$E(Um) = \sum_{i=1}^{m} E(1Aj)$$

$$= \sum_{i=1}^{m} Prob\{Aj\}$$

$$= m\left(1 - \frac{1}{m}\right)^{np}$$

$$= m(e^{\frac{-np}{m}})$$

$$E(Vm) = E(Um)/m$$

$$= e^{\frac{-np}{m}}$$

$$n = (approximately)\ mlnE(Vm)/p$$

$$\bar{n} = -mln(Vm)/p$$

Similarly we can also calculate variance for the new algorithim which has sampling included.

Variance

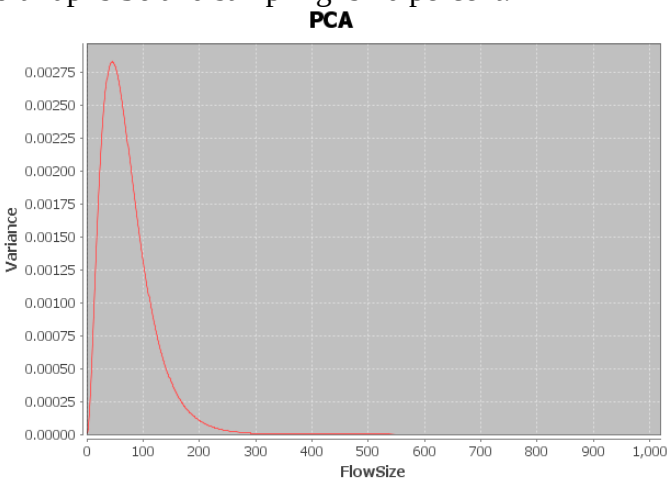
$$\mathbf{Var}(V_n) = \frac{1}{m} e^{-t}(1 - (1 + t)e^{-t})$$

Here m is the size of our bitmap and t is the load factor. We define t as n/m that is flow size/ size of bitmap, now since we have introduced sampling into the algorithm our load factor changes to :

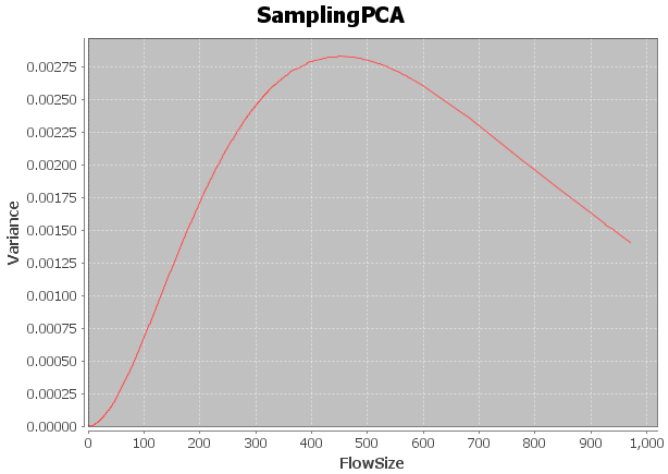
load factor **(t) = p *(n/m)** - load factor

If we now plot a graph of old algorithm and new sampling algorithm for variance at the point where old algorithm was giving wrong results we get following

Here graph 1 is old graph when size of bitmap is 36 and graph two is the new sampling algorithm when bitmap is 36 and sampling is 10 percent.



Graph1



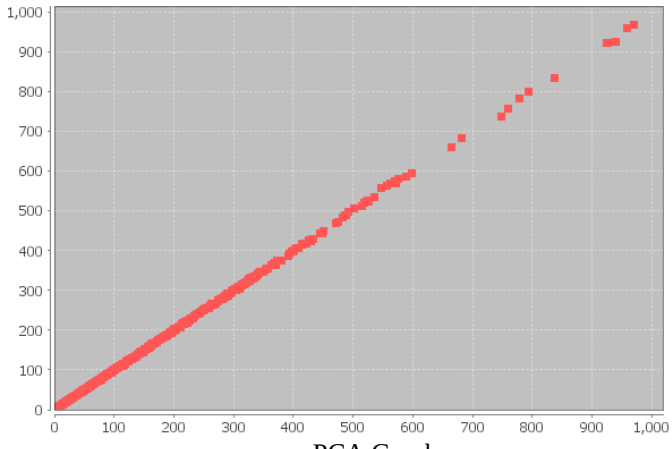
Graph2

IV. EXPRTIMENTAL EVALUATION

To showcase that the sampling algorithm I first have to find a point where normal PCA starts to give wrong results. As explained above when measuring a thousand element once bitmap reduces to 200 , the output that is the cardinality starts to skew. Once we get skew results we start applying the sampling rate and re run to get the estimated cardinality again. For experimental evaluation all implementations have been done in java and in order to represent graphs we have used jFreeCharts.

A. bitmap size > 200 no sampling used (PCA)

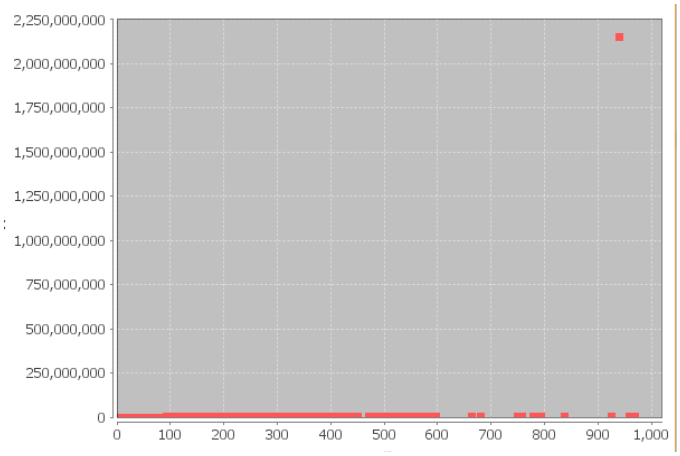
Below graph is run on original PCA when our bit vector size is greater than 200. It gives good results. No sampling is used in below graph



PCA Graph

B. bitmap size = 200 no sampling used (PCA)

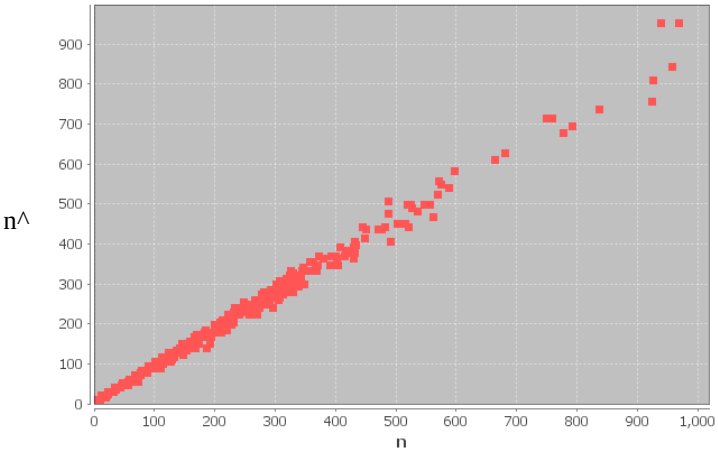
Now as soon as we reduce the size of bitmap to 200 results become skew and PCA original is not able to estimate the cardinalities correctly.



PCA Graph

C. bitmap size = 200 sampling rate of 20 percent

This graph is for 200 bit vector bit with 20 percent sampling applied, we can see that results start to give correct estimated cardinalities when sampling is used.



As we had discussed earlier with each experimental result there is an error rate which we have to compromise. Below table shows a relation between sampling rate and error perecentage.

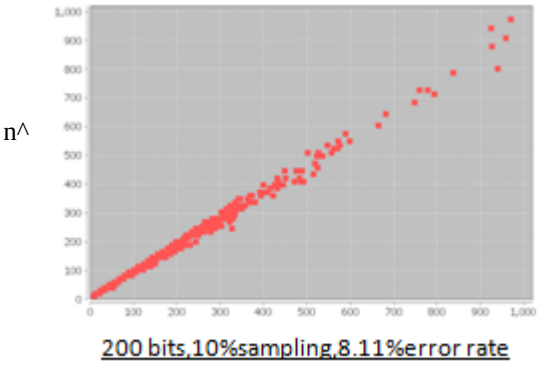
Table 1

Sampling Rate	% Error
90	0.096
80	0.158
70	0.26
60	0.57
50	1.55
40	1.85
30	2.87
20	4.86

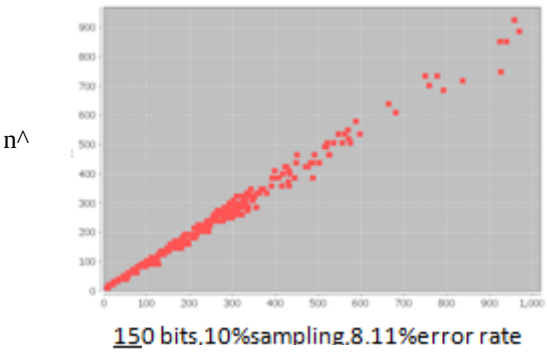
As we can see from the table error rate increases as we reduce the sampling rate. This happens because as we increase the sampling rate we decrease the rate at which we extrapolate the data and as we decrease the sampling rate we increase the percentage of data that we extrapolate. We need to be clear of the fact that above table is created for 200 bit long bitmap. A 200 long bitmap does not result into acceptable results but when we use any of the above sampling rate it results in acceptable results with the error rate mentioned above.

It is interesting to explore the limit to which we can get correct results with a fixed sampling results if we keep on reducing the number of bits. Our aim of experiment is twofold. First is to get acceptable results by using sampling rate on those value of size of bitmap where normal PCA algorithm fails to give correct result, second to further reduce the memory consumption by decreasing the size of bitmap and selecting a sampling rate which gives correct results.

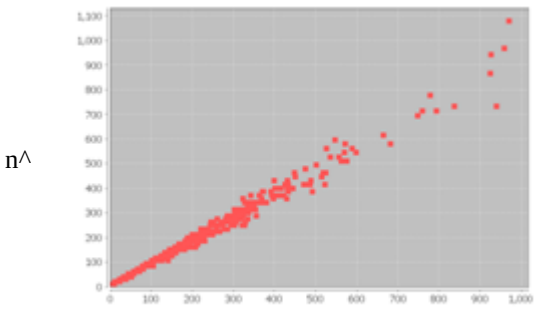
Below are graphs which were plotted for different bitmap size till we get an unacceptable error rate.

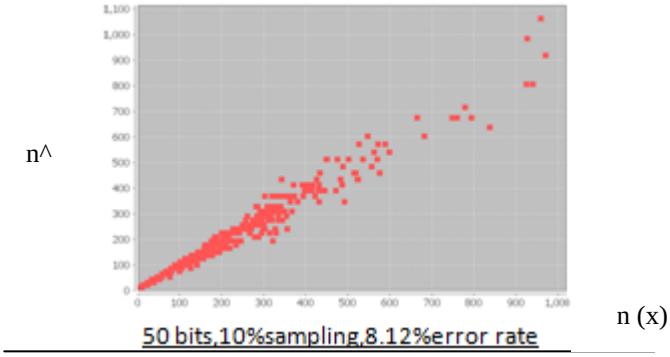


n (original) x axis



n (original) x axis





It is Interesting to see how much space can we save by using minimum sampling rate and maximazing the extrapolation and noting the error rate which we get. We can see that graph starts to loose its narrow property from lower values of cardinality as we go up to higher cardinality flows.

V. Conclusion

Our conclusion is multifolds here, In order to capture results for conclusion I have used the same graphs and analysed them by comparing them against the original PCA. The results can be drawn into different vectors, my areas of analysis are memory consumption and correct results.

A. We were successfully able to introduce and test sampling rate in PCA algorithm and test out using different bitmap size by plotting a scatter graph

B. We have seen that we are able to save memory as we select an appropriate sampling rate and acceptable error rate

C. We can make the PCA algorithim work at 200 bits with 20 percent sampling thus an improvement over the orignal PCA algorithm

VI. References

[1] Yu, C. T., LAM, K., SIU, M. K., AND OZSOYOGLU, M. Performance analysis of three related assignment problems. In Proceedings of the International Conference on Management of Data,

[2] YOUSSEFI, K., AND WONG, E. Query processing in a relational database management system.

[3] JOHNSON, N. L., AND KOTZ, S. Urn Models and Their Application. Wiley, New York, 1977.I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[4] ROWE, N. C. Antisampling for estimation: An overview. IEEE Trans. Softw. Eng. SE-II, 10 (Oct. 1985), 1081-1091.

[5] Wenjia Fang and Larry Peterson. Inter-as traffic patterns and their implications. In Proceedings of IEEE GLOBECOM, December 1999.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. ACM SIGCOMM CCR, 32(3):62–73, July 2002.

[8] Bitmap Algorithms for Counting Active Flows on High Speed Links Cristian Estan, George Varghese, Mike Fisk

[9] A Linear-Time Probabilistic Counting Algorithm for Database Applications KYU-YOUNG WHANG Korea Advanced Institute of Science and Technology BRAD T. VANDER-ZANDEN Cornell University and HOWARD M. TAYLOR University of Delaware