Dynamic Count-min sketch

Haja Mohideen J and Shaalini S

PSG College of Technology

Abstract. While analyzing the big data, the information about the unique items, occurrence of each item are need to be kept tracked. Hash tables as usually used for this purpose. When the data becomes too large, it can't be fitted into a single hash table. Thus, probabilistic data structures are used for streaming or huge data. These data structures ignore collisions and errors are controlled under a certain specified threshold. Dynamic Count-min Sketch is a probabilistic data structure that is used to store and estimate the frequency for any item in the given multiset.

Keywords: Count-min Sketch(CMS), Bloom Filters(BF), Probabilistic data structure, Streaming data, Dynamic allocation.

1 Introduction

1.1 Background and Motivation

Given a multiset, estimating the frequency of each item is a critical problem in data stream applications. A multiset refers to a set in which each item can appear multiple times. In scenarios such as real-time IP traffic, graph streams, web clicks and crawls, sensor database, and natural language processing (NLP), the massive data are often organized as high-speed streams, requiring servers to record stream information in real time. Due to the high speed of data streams, it is often impractical to achieve accurate recording and estimating of item frequencies. Therefore, estimation of item frequencies by probabilistic data structures becomes popular and gains wide acceptances. Sketches are initially designed for the estimation of item frequencies in data streams, and now have been applied to many more fields, such as sparse approximation in compressed sensing, natural language processing, data graph, and more. We proposed a new probabilistic data structure in this paper, Dynamic Count-min sketch that allocates dynamic memory for the streaming data. This overcomes the disadvantage of static memory allocation in the existing data structures thereby maintaining the accuracy over any large data. The design and implementation of Dynamic Count-min Sketch is discussed in this paper and also, compared with different existing data structures.

1.2 Proposed Solution

In this paper, we propose a sketch framework, namely the Dynamic Count-min sketch, as it employs dynamically allocated Count-min sketch data structure[5].

The key idea of our Dynamic Count-min sketch framework is to automatically increase the size of the Count min-sketch according to the current frequency of the incoming item,which additionally includes a bloom filter that results in high accuracy for estimation. Our proposed enlarging strategy can significantly improve the accuracy. Considering the significance of estimation speed in high speed data streams, we further proposed another approach to increase the throughput for the given estimation algorithm to accelerate the estimation speed, while keeping the same accuracy.

To verify the performance of our sketch framework, we compared dynamic count-min sketch with pyramid sketch framework as it achieves the highest accuracy and highest estimation speed at the same time.

2 Literature Review

The issue of estimation of the item frequency in a multiset is a fundamental problem in databases. The solutions can be divided into the following three categories: sketches[4][12][5], Bloom filter variants[1][10], and counter variants[3][2].

Sketches: Typical sketches include CM sketches[5], CU sketches[6], Count sketches[7], Augmented sketches[9], and pyramid sketch[12]. A comprehensive survey about sketch algorithms is provided in the literature. A CM sketch consists of d arrays, $[A_1...A_d]$, and each array consists of w counters. There are d hash functions, $h_1(.)...h_d(.)$ $(1 \leq h(.) \leq w)$. When inserting an item e, the CM sketch increments all the d mapped counters $A_1[h_1(e)]...$ $A_d[h_d(e)]$ by 1. When querying an item e, the CM sketch reports the minimal one among the d mapped counters. The CU sketch is similar to the CM sketch except that it only increments the smallest counters among the d mapped counters for each insertion. The Count sketch[7] is also similar to the CM sketch except that each array is associated with two hash functions. The Augmented sketch targets at improving accuracy by using one additional filter to dynamically capture hot items, suffering from complexities, slow update and query speed. The Augmented sketch adds an additional filter (which is actually a queue with k items) to an existing sketch T.The Augmented sketch is very accurate only for those items in the filter. The authors focus on querying the hot items by using query sets sampled from the whole multiset. That is why their experimental results of the Augmented sketch significantly outperform the CM sketch. The pyramid sketch consists of d counters with a single hash function that updates the value of an item dynamically while the size increases. The sketch contains parent and child counters with an indicator variable that increases as the sketch grows. Among these sketches, pyramid sketch gives better accuracy for skewed data sets.

Bloom filter variants: The second kind of solutions is based on Bloom filter. A standard Bloom filter can tell whether an item belongs to a set or not, but cannot estimate its frequency. Counting Bloom filters can be used to estimate the

frequency of items in a multiset. CBF is quite similar to the CM sketches, except that CBF uses only one array. Several improvements based on CBF have been proposed, such as the Spectral Bloom Filter (SBF)[10], the Dynamic Count Filter (DCF)[1] and the Shifting Bloom filter[11], and they all can store frequencies of items.

Summary: Although there are various algorithms for frequency estimation of multisets, no existing sketch can achieve high accuracy and one memory access per estimation for an item.

3 Dynamic Count-min sketch framework

In this section, we present the Dynamic count-min sketch framework. It supports 2 methods Update and Estimate. Update is used to update the frequency of the item in DCMS. Estimate is to estimate the frequency of an item in DCMS. The structure and the algorithms of Update and Estimate is discussed in this section.

3.1 Data Structure

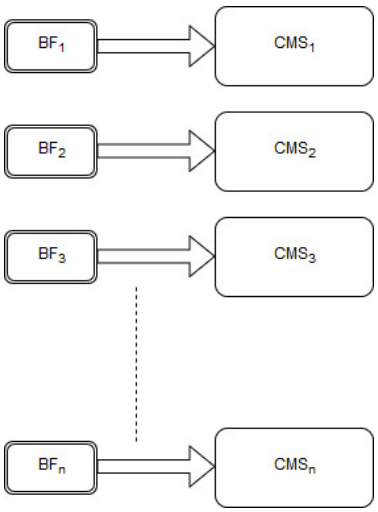

Fig. 1. DCMS Structure

The structure of Dynamic Count-min sketch is shown in the Fig.1. Initially, Dynamic Count-min sketch consists of a bloom filter(BF_1) and a Count-min sketch (CMS_1) with hyperparameters as b hash functions for bloom filter and width w, depth d and maximum count allowed maxCount for Count-min sketch.

As the no. of items to be updated gets increased, the number of CMS and BF gets increased after some m updation. In general, Dynamic Count-min sketch consists of CMS_1, CMS_2 .... CMS_n Count-min sketches and corresponding Bloom filters as BF_1, BF_2, ... BF_n at the end of whole update.This makes the dynamic count-min sketch to maintain better accuracy even with scaling of input data. This is possible because the collision gets decreased for increase in number of count-min sketch.

3.2 Update

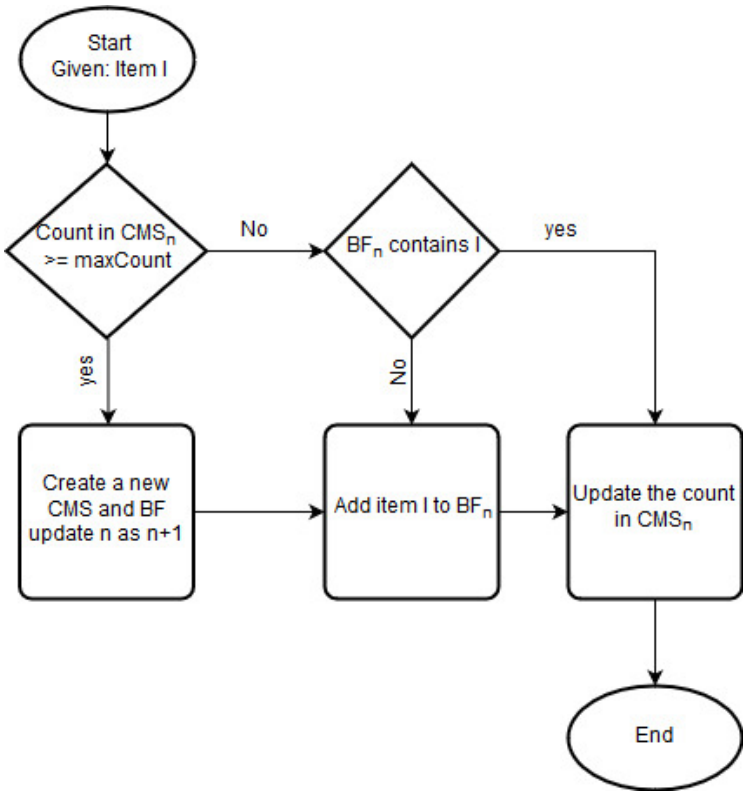

Fig. 2. Update

The flowchart of update operation is shown in fig.2. The detailed explanation of this flowchart is dicussed here.

When updating an item I with count c, initially we would update the count c to the recently initialized count-min sketch CMS_n along with the corresponding bloom filter BF_n. For updating the item I, we first check if the current CMS

have reached the maximum count MaxCount that it can hold. If it is not reached then the item I is updated in BF_n and CMS_n with how much ever that CMS can hold (i.e maxCount - currentCount in CMS_n of c) and the remaining part of c is updated by creating a new CMS and a new BF updating the remaining count in the new CMS (i.e c-(maxCount - currentCount)) and updating the membership of new BF with item I else a new CMS is created and count of item I is updated in similar manner. In this way, we dynamically assign an appropriate increment to each of the CMS, which results in the collision reduction, so as to increase the accuracy.

Example: The item "abcdefghij" need to be updated with count 5. There are 15 (i.e n = 15) CMS created so far and maxCount is 10. The CMS_{15} contains 8 counts.

Steps:

1. Go to the BF_{15} and the corresponding CMS_{15}.
2. CMS_{15} should increment the count of the given item to 2. BF_{15} is also updated with the membership of this item.
3. the remaining count 3 is updated by creating a new CMS_{16} and a new BF_{16} and the remaining count of 3 is updated here.

3.3 Estimate

The flowchart of the estimate operation is given in the fig 3 and the detailed explanation is given below.

When estimating an item I, each of the count-min sketch estimations of this item I are needed to be conquered. Initially, the estimation of I, c is marked to be 0. Then, the item I is queried with the bloom filter BF_i, if BF_i says yes then it indicates that possibility of the item I being updated in the coincided count-min sketch CMS_i is high. So, that CMS_i is need to be conquered. c is increment with the estimation of e from that count-min sketch CMS_i . This process is repeated till i = n. Finally, the estimated value of item I is returned.

Example: The item "abcdefghij" need to be estimated. There are 3 (i.e n = 3) CMS created so far and item "abcdefghij" was initially updated in CMS_1 and CMS_3 with counts 3, 4 respectively.

Steps:

1. Initialize c = 0.
2. BF_1 is queried for membership of the given item. It says Yes, so c = c + estimate of CMS_1 of the given item.
3. BF_2 is queried it says No, so this step is skipped.
4. BF_3 is queried it says Yes, c = c + estimate of given item.
5. c is returned.

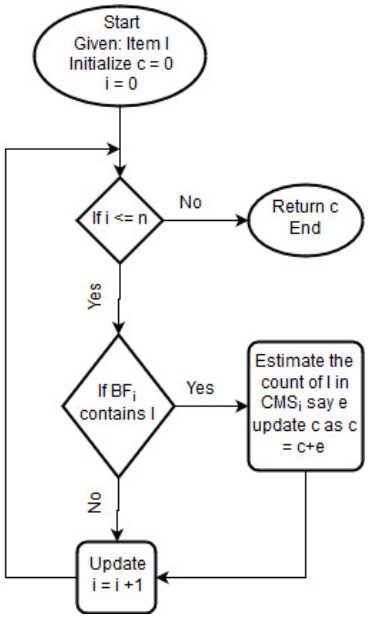

Fig. 3. Estimate

4 Parallel Implementation

Due to physical limitations, the individual computer processor has largely reached the upper ceiling for speed with current designs. So, hardware makers added more processors to the motherboard (parallel CPU cores, running at the same speed). Data collection has gotten exponentially bigger, due to cheap devices that can collect specific data (such as temperature, sound, speed. . .).To process this data in a more efficient way, newer programming methods were needed. A cluster of computing processes is similar to a group of workers. A team can work better and more efficiently than a single worker. They pool resources. This means they share information, break down the tasks and collect updates and outputs to come up with a single set of results.As Parallel Methods have already been discussed in [5]. it is easy for us to incorporate these ideas in DCMS also easily. These kind of parallel processing implementation breaks our single iteration that estimates the values in each of the CMS. So, our accuracy is maintained along with the increase in throughput.

5 Performance Evaluation

5.1 Experimental Setup

Data set: Real IP-Trace Streams:We obtain the real IP traces from the main gateway at our campus. The IP traces consist of flows, and each flow is identified

by its five-tuple: source IP address, destination IP address, source port, destination port, and protocol type. The estimation of item frequency corresponds to the estimation of number of packets in a flow. We divide 10M*10 packets into 10 data sets, and build a sketch with 1MB memory for each data set. Each data set includes around 1M flows. The flow characteristics in these data sets are similar. Take the first data set as an example. The flow size ranges from 1 to 25,429 with a mean of 9.27 and a variance of 11,361. Note that 41.8% flows only have one packet. The length of items in each experimental data set is 22 ~46 bytes.

5.2 Implementation and Results

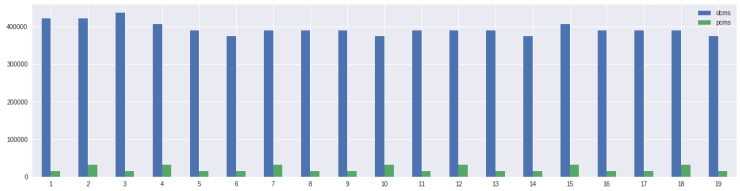

Fig. 4. DCM with 50 CMS Vs PCM

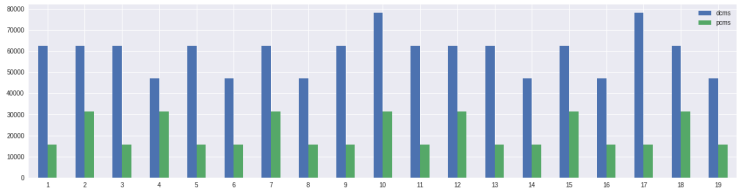

Fig. 5. DCM with 10 CMS Vs PCM

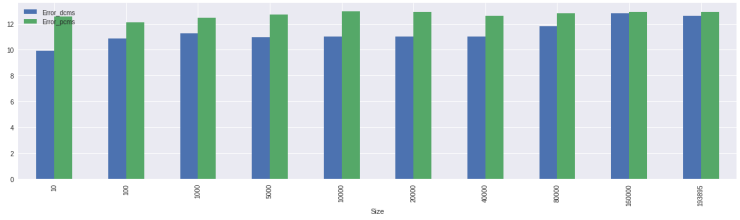

Fig. 6. MAE of DCM Vs PCM

We have implemented the dynamic count-min sketch and pyramid sketch in C++[8]. We apply our dynamic framework to these sketches, and the results are denoted as PCM and DCM. The hash functions used in the sketches are implemented from the 32-bit or 64-bit Bob Hash with different initial seeds.

Fig.4 interprets the time taken for the completion of execution(y-axis) of our DCM framework(blue) with the existing PCM framework(green) for the increase

in scalability(x-axis). From this, it is clear that as the number of DMS increases, the time taken for execution of operation also gets increased. We compared it with the optimized parameters of PCM to the DCM sketch, which consists of 50 CMS.

Later, we reduced the number of CMS by increasing the maxCount and the results have been shown in Fig.5. From this, it is clear that the time taken for its execution decreases gradually as the number of CMS decreases. We can still improve by executing it parallely.

In Fig.6, Mean Absolute Error(MAE)(y-axis) have been calculated for the optimized hyperparameters of PCM(green) and DCM(blue)(x-axis). From Figure, we could interpret that our DCM sketch gives better performance than the existing PCM for different scales of data.

5.3 Computation Platform

We performed all the experiments on a machine with 4-core CPUs (Intel(R) Core(TM) i7-6700HQ CPU @ 2.60 GHz) and 16 GB total DRAM memory. CPU has three levels of cache memory: two 32KB L1 caches (one is a data cache and the other is an instruction cache) for each core, one 256KB L2 cache for each core, and one 15MB L3 cache shared by all cores.

6 Conclusion

Sketches have been applied to various fields. In this paper, we propose a sketch framework - dynamic count min sketch, to significantly improve the update speed and accuracy. We compared our framework with pyramid sketch. Experimental results shows that our framework significantly improves both accuracy and speed. We believe our framework can be applied to many more sketches.

7 References

1. Aguilar-Saborit, J., Trancoso, P., Muntés-Mulero, V., Larriba-Pey, J.: Dynamic count filters. SIGMOD Record 35(1), 26–32 (2006), https://doi.org/10.1145/1121995.1122000
2. Aguilar-Saborit, J., Trancoso, P., Muntés-Mulero, V., Larriba-Pey, J.: Dynamic count filters. SIGMOD Record 35(1), 26–32 (2006), https://doi.org/10.1145/1121995.1122000
3. Bonomi, F., Mitzenmacher, M., Panigrahy, R., Singh, S., Varghese, G.: An improved construction for counting bloom filters. In: Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings. pp. 684–695 (2006), https://doi.org/10.1007/11841036_61
4. Cormode, G.: Sketch techniques for approximate query processing. Foundations and Trends in Databases. NOW publishers (2011)
5. Cormode, G.: Count-min sketch. In: Encyclopedia of Database Systems, Second Edition (2018), https://doi.org/10.1007/978-1-4614-8265-9_87

6. Estan, C., Savage, S., Varghese, G.: Automated measurement of high volume traffic clusters. Proceedings of the second ACM SIGCOMM Workshop on Internet measurment - IMW 02 (2002)
7. Estan, C., Savage, S., Varghese, G.: Automated measurement of high volume traffic clusters. Proceedings of the second ACM SIGCOMM Workshop on Internet measurment - IMW 02 (2002)
8. Fazilhaja: fazilhaja/dynamiccountminsketch (Apr 2019), https://github.com/fazilhaja/DynamicCountMinSketch
9. Roy, P., Khan, A., Alonso, G.: Augmented sketch: Faster and more accurate stream processing. In: Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016. pp. 1449–1463 (2016), https://doi.org/10.1145/2882903.2882948
10. Tarkoma, S., Rothenberg, C.E., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. IEEE Communications Surveys and Tutorials 14(1), 131–155 (2012), https://doi.org/10.1109/SURV.2011.031611.00024
11. Yang, T., Liu, A.X., Shahzad, M., Zhong, Y., Fu, Q., Li, Z., Xie, G., Li, X.: A shifting bloom filter framework for set queries. PVLDB 9(5), 408–419 (2016), http://www.vldb.org/pvldb/vol9/p408-yang.pdf
12. Yang, T., Zhou, Y., Jin, H., Chen, S., Li, X.: Pyramid sketch: a sketch framework for frequency estimation of data streams. PVLDB 10(11), 1442–1453 (2017), http://www.vldb.org/pvldb/vol10/p1442-yang.pdf