2019 Seventh International Conference on Advanced Cloud and Big Data (CBD)

CBFSketch: A scalable sketch framework for high speed network

Haiting Zhu*, Yuan Zhang*, Lu Zhang†, Gaofeng He*‡, Linfeng Liu§
*School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China
†School of Information Engineering, Nanjing University of Finance and Economics, Nanjing, China
‡Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education
§School of Computer Science & Technology, Nanjing University of Posts and Telecommunications, Nanjing, China
*Corresponding author: hegaofeng@njupt.edu.cn

***Abstract*—Sketch is a compact data structure used to summarize data streams. It has high measurement accuracy and uses less space than traditional measurement methods. Thus it is widely used in the measurement of network traffic. Currently, there are some typical sketches: Count-Min Sketch, CU Sketch, and Count Sketch, but they still waste memory when storing mice flows. To further improve memory space utilization, we propose a new sketch framework CBFSketch, which is combined with Count-Min Sketch and Bloom Filter. In the new framework, the sketch is created dynamically and the memory space is adjusted timely according to the traffic size. Our experiments show that the space utilization can be significantly improved, and in the case of using the same memory space, the average relative error is reduced by about 20% to 50% compared with other sketches.**

***Keywords*—Traffic measurement; Data stream technology; Sketch; Bloom Filter; Space utilization**

I. Introduction

A. Background and Motivation

Network measurement and analysis is very important for the network management. It can help network administrators to better understand the status of the network and make the right decisions. It is also helps to detect anomalies and attacks in the network.

In recent years, network traffic measurement has been widely used in network accounting, traffic engineering, network security and other fields. However, there are certain difficulties in the measurement, which are mainly due to the continuous and massive network traffic. At present, there are two main types of high-speed network traffic measurement schemes, one is the sampling technology, and the other is the data stream technology. Sampling technology refers to the collection and processing of some representative network traffic. This method introduces the small system load with a relatively large error. Sampling methods mainly include group sampling (such as BSS [1], EBS [2]) and stream sampling (such as ANF [3]). Data stream technology processes all network traffic data. It not only has low demand for storage resources, but also maintains certain accuracy.

The data stream method can be used to measure network traffic statistics on high-speed links, such as the total number of active flows, large stream identification, distribution of flow size, node connectivity and entropy estimation. The existing data stream methods mainly include Bitmap [4], Bloom filter [5] and Count-Min Sketch [6].

Count-Min Sketch [6] is a data structure based on sketch. It can summarize the data stream and measure the data stream. It not only has great flexibility and other powerful features, but also has little impact on the system. Sketch is especially suitable for data streaming solutions. Count-Min Sketch was proposed by Gormode G to summarize the data flow in the network and support a series of query operations on the data stream. Count-Min Sketch is a two-dimensional array structure, each row has a hash function and several counters. It needs to calculate the hash value and find the mapped counter in each row. Then the flow is stored in the mapped counter. It supports point queries, range queries, and inner product queries for data streams. It also can detect the heavy hitters accurately. In addition, there are other the sketch-based data structures, such as Count Sketch [7], Deltoid [8]. Sketch can be used to detect flood attacks of DDoS in the network [9]. Sketch can also be used for network security, sparse approximation in compressed sensing [10], natural language processing [11], data graph [12], machine learning [13] and so on.

In actual network traffic, the distribution of flow size is skewed: a few flows have large sizes (i.e., elephant flows) while most flows only have small sizes (i.e., mice flows). Using sketch for network measurement requires storage of all data streams in the network. The larger counters are required to store elephant flows accurately, which requires a large memory space. However, there are a few elephant flows. So most of the memory space is used to store the mice flows. However, the mice flows does not require such a large counter space. As a result, a large amount of space is wasted, resulting in low space utilization.

B. The Proposed Solution

In this paper, we proposed a sketch-based data framework: CBFSketch (Counting Bloom Filter Sketch). The main idea of CBFSketch is to combine several small size Count-Min Sketches and a Counting Bloom Filter [14]. In order to take advantage of the memory space, we dynamically creates multilayer sketches.

In CBFSketch, there is one initial sketch. If the counter of this sketch overflowed, the next sketch is created to store the

978-1-7281-5141-0/19/$31.00 ©2019 IEEE
DOI 10.1109/CBD.2019.00070

overflow items. In this way, CBFSketch dynamically creates sketches based on the size of the flow. Thus it avoids the waste of memory space and improves the efficiency of space usage.

II. Related Work

In order to measure high-speed network traffic, the existing main methods are sampling technology, Bitmap, sketch-based data structure and Bloom Filter-based data structure.

The typical algorithms are Frequent [15], Lossy Counting [16], and Space-Saving [17]. Taking Space-Saving as an example, recognizing that it is infeasible to count the sizes of all flows, Space-Saving counts only the sizes of some flows in a data structure called Stream-Summary, which incurs O(1) overhead to search or update a flow, or find the smallest flow. The selection of which flow to store in the summary is rather simple: For each arrival packet, if its flow ID is not in the summary, the flow will be admitted into the summary, replacing the smallest existing flow. The initial size of the new flow is set to $n_{min}+1$, where n_{min} is the smallest flow size in the summary before replacement. Therefore, later incoming mice flows will be largely over-estimated, which is drastically inaccurate. In the end, the largest k flows in the summary will be reported.

Bitmap [4] is a simple data structure that maps a domain to a bit array. The direct Bitmap is a stream number estimation algorithm that uses a hash function to map a flow identifier to a bit in a Bitmap and sets the bit to 1 which is used as an estimate of the number of streams.

Sketch was originally used to summarize data streams and record the frequency of data stream. The typical sketches include Count-Min Sketch [6], Count Sketch [7], CU Sketch [18], and Deltoid [8]. A Count-Min Sketch contains d arrays, $A_1, A_2, ..., A_d$, and each array contains w counters. At the same time, Count-Min Sketch also contains d hash functions, $h_1(.), h_2(.), ..., h_d(.)$, $(1 \leq h(.) \leq w)$. When it wants to insert an item, it performs d hash calculation on the key (source IP address, five tuple, etc.) of the item to determine the mapped counter in each row, $A_1[h_1(e)], A_2[h_2(e)], ..., A_d[h_d(e)]$. Then it adds the value to the counter. When it wants to query the value of an item, it performs hash calculations on the key of the item. Then it can find all mapped counters and select the smallest value as the estimated value of the item. The data structure of Count Sketch is similar to Count-Min Sketch. The difference between them is that each array of Count Sketch is related to two hash functions. When it wants to query an item, Count Sketch selects the median as the estimated value. Deltoid is based on Count-Min Sketch and used to detect traffic changes in the network. Deltoid can detect traffic that has changed dramatically.

Bloom Filter [5] is a simple and efficient data structure. The standard Bloom Filter is a binary vector data structure. It can judge whether an item belongs to a set, but cannot estimate its frequency. CBF [14] (Counting Bloom Filter) is a new data structure based on Bloom Filter improvement, which can be used to estimate the frequency of the items. When it wants to query the item, it will check all mapped counters. If the value of all counters is non-zero, it means the item exists; otherwise, it means the item does not exist. In addition, the improved structure based on Bloom Filter are SPF [19] (Spectral Bloom Filter), DCF [20] (Dynamic Counter Filter) and Shifting Bloom Filter [21], which can be used to store the frequency of the items.

The current sketch-based data stream structure can store and summarize data streams and support to query the frequency of the item. However, some important issues remain in it. Firstly, all flows are mapped to the sketch. Each counter may store multiple flows and record the sum of size of all flows. It can cause the over-estimation problem. If the mice flow and the elephant flow stored in the same counter, the mice flow will be over-estimated. Secondly, in order to store all the data stream in the sketch and ensure a certain accuracy, large memory space needs to be allocated to the counter of the sketch to ensure that elephant flows can be stored in the sketch. However, most of the actual network flows are mice flows. When it stores the mice flows, the high bits in the counter is almost unnecessary to use, resulting in space waste and low space utilization.

III. CBFSketch

In this section, we introduce the data structure of CBFSketch (Counting Bloom Filter Sketch) and the operations it supports (insert, query). We illustrate how it can dynamically create sketches to improve memory space utilization.

A. CBFSketch Structure

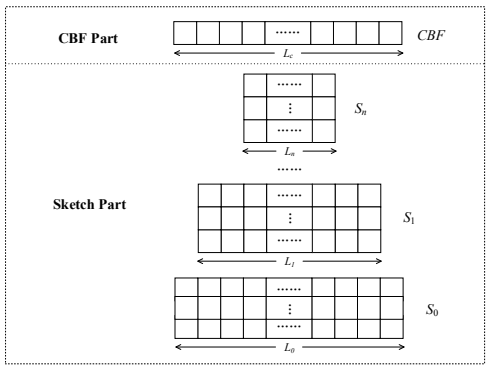


Fig. 1. The structure of CBFSketch.

To overcome the issue of the low space utilization, we use several small size sketches to store the data stream. CBFSketch dynamically creates small size sketches and uses these sketches to store data streams. In the beginning, CBFSketch generates a single initial sketch. If the counter of this sketch overflows, we create the next sketch and store the overflow item in it. The higher layer sketch is smaller than the lower sketch. Due to the small amount of elephant flows, there are not too many overflowing flows, and we can continue to reduce the size of higher layer sketches.

CBFSketch is a data structure mainly based on sketch. Its structure is shown in Fig. 1. It consists of two parts, the sketch part and the CBF part. The Sketch part contains $n+1(n \geq 0)$ Count-Min Sketch, $S_0, S_1, ..., S_n$. The CBF part contains a Counting Bloom Filter (CBF).

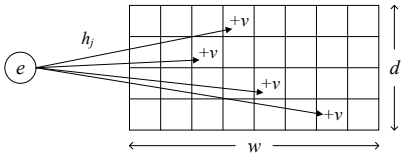


Fig. 2. The structure and update process of Count-Min Sketch.

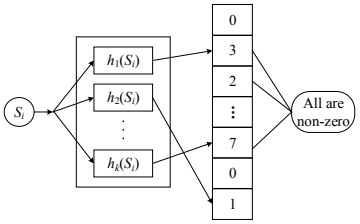
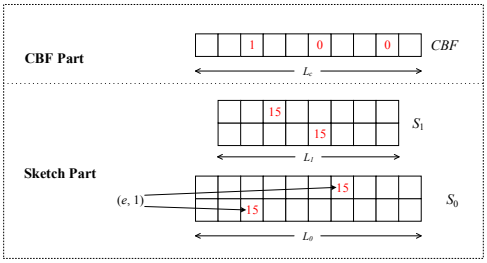


Fig. 3. The structure of Counting Bloom Filter.

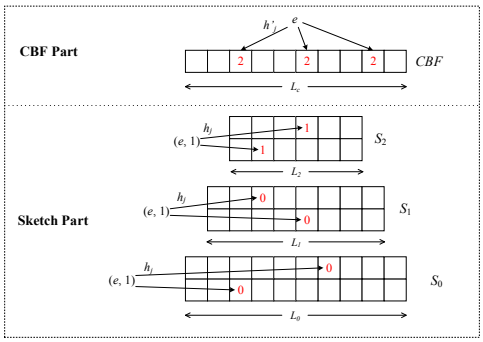
The structure of each Count-Min Sketch is shown in Fig. 2. Each sketch contains d arrays, $A_1, A_2, ..., A_d$, and each array contains w counters. At the same time, Count-Min Sketch also contains d hash functions, $h_1(.), h_2(.), ..., h_d(.)(\ 1 \leq h(.) \leq w)$. In CBFSketch, the w of each Count-Min Sketch are $L_0, L_1, ..., L_n$, and the counter size of each Count-Min Sketch are $k_0, k_1, ..., k_n$. The update process of Count-Min Sketch is shown in Fig. 2. When it wants to inserts an item (e, v) in Count-Min Sketch, it first performs hash calculations on e, $A_1[h_1(e)], A_2[h_2(e)], ..., Ad[hd(e)]$, which indicate the mapped counters in each row, and then the value v corresponding to the item e is added to the counter. When it wants to query the estimated value of an item, it selects the minimum value of all counters as the estimated value. The structure of the Counting Bloom Filter is shown in Fig. 3. It is improved by the standard Bloom Filter. CBF expands each bit of the Bloom Filter into a counter. It is similar in structure to Count-Min Sketch, but the difference between them is that Count-Min Sketch is a two-dimensional array structure with k hash functions while CBF is a one-dimensional array with a length of L_c. When an item is to be inserted into the CBF, it performs k hash calculations on the item firstly and increase the mapped counter by one. When it wants to query the item, its check all mapped counters. Then it selects the minimum value as the number of the sketches that the item uses.

B. Insert

At initialization, CBFSketch generates only one Count-Min Sketch (S_0) and one CBF, and all have an initial value of 0. When a new item (e, v) arrives (e is the key of the item, v is the value of the item. If we want to record the size of data packets, v is the length of the item; if we want to record the number of occurrences of the item, v is 1.), the process of inserting the item is shown in Fig. 4:



(a) Before the item insert



(b) After the item insert

Fig. 4. An example of inserting item.

1) It hashes the key e of the item and calculates the mapped counters in S_0. Then it finds the counters with the minimum value and adds the value v to the counters. If there are several counters having the minimum value, it adds v to all the values of these counters.
2) If the value of the counter exceeds the maximum value of the counter in S_i after adding v and S_{i+1} does not exist, it triggers creation of the next Count-Min Sketch (S_{i+1}). Then it sets the value of all the minimum mapped counters in S_i to $c_i + v - 2^{k_{i-1}}$, and add 1 to S_{i+1} in the mapped counter, where c_i is the current value of this counter. Then it performs hash calculations on e in CBF and add 1 to the minimum counters among the mapped counter in CBF. The value (dep) of the counter of CBF means that the item is stored from S_0 to S_{dep}.
3) If the value of the counter exceeds the maximum value of the counter in S_i after adding v and S_{i+1} exists, S_{i+1} and CBF update the item directly the same as in step 2). If overflow still happens during the updating, repeat steps 2) or 3) to create the next Count-Min Sketch and update the value of the mapped counters in CBF until no overflow occurs.

During the process of inserting items in the sketch part

and CBF part, it only adds values to the minimum mapped counters. It can reduce the error caused by the conflicts. In the traditional Count-Min Sketch, it adds values to the all mapped counters. When a conflict occurs, adding the value to all mapped counters continuously will increase the error of the counter. Fig. 4 shows an example of inserting an item. Before an item arrives, the state of CBFSketch is shown in Fig. 4(a). At present, there are two sketches (S_0 and S_1) and one CBF. The size of each sketch and CBF are 4 bits. Firstly, the item (e, 1) arrives, the minimum values of the mapped counters in S_0 are 15. 15 is maximum value of the counters. After adding 1 to these counters, they overflow. Then setting them to 0, and find the mapped counter in CBF. The current value of the mapped counters are 1, 0, 0 respectively, and the minimum value are added by 1. Secondly, adding 1 to the minimum mapped counters of S_1. The counters also overflow. Then setting them to 0. And the higher layer sketch does not exist. So CBFSketch creates the higher layer sketch (S_1) and updates S_1 and CBF according to the above process. The state of CBFSketch is shown in Fig. 4(b).

C. Query

CBFSketch supports the operation of querying the value of the items in it.

1) Firstly, CBFSketch performs k hash operations on the item (e, v) in the CBF to find the k counters corresponding to the item. Then it selects the minimum value as the depth and records it as dep.
2) Secondly, in $S_{dep}, ..., S_1, S_0$, it performs hash operations to finds the mapped counters in each sketch, and selects the counter with the minimum value as the estimated value, which are respectively recorded as $V_{dep}, ..., V_1, V_0$.
3) Finally, it uses the the equation (1) to get final estimate of the item.

$$V_{est} = V_0 + \sum_{i=1}^{dep} (V_i \cdot 2^{\sum_{j=0}^{i-1} k_j}) \tag{1}$$

D. Performance Analysis

Inserting and querying in a large sketch takes longer times than a small sketch. Therefore, the accumulation of the insertion and query time in several small sketches is comparable to that of a large sketch. So in terms of the throughput of inserting and querying, CBFSketch is basically the same as Count-Min Sketch.

CBFSketch can quickly store and summarize data streams and support queries for data streams. When it needs to store data streams, sketches are dynamically created based on the size of the data stream to improve space utilization and save space. Each time a higher layer sketch is created, it is smaller than the previous sketch. The size of each Count-Min Sketch is $M_0, M_1, ..., M_n$. The sizes of sketches decrease linearly to make the space contronllable. The size of CBF is M_c. The size relationship between each sketch is as follows:

$$M_{i+1} = r \cdot M_i (0 < r < 1) \tag{2}$$

The total amount of memory space used by the sketch section is:

$$M = M_0 \cdot \frac{1 - r^{n+1}}{1 - r} + M_c \ (0 < r < 1) \tag{3}$$

$$\lim_{n \to \infty} (M_0 \cdot \frac{1 - r^{n+1}}{1 - r} + M_c) = \frac{M_0}{1 - r} + M_c \tag{4}$$

Therefore, the space used by the sketch part is limited, and there is an upper limit of the space used. The CBF part is a one-dimensional array structure of the Counting Bloom Filter, and the space used is a fixed size. The total memory space of CBFSketch is shown in equation (3). Therefore, CBFSketch has an upper limit of the amount of memory space used, and the size of the space is controllable as shown in equation (4).

In terms of accuracy, the accuracy of Count-Min Sketch is related to its parameters w, d, where $w = \lceil e/\varepsilon \rceil$ (e is Euler number), $d = \lceil \ln(1/\delta) \rceil$. In Count-Min Sketch, the point query value is used to query the corresponding value. The error of the point query and the probability of occurrence of the error are:

$$P(\hat{a}_i \leq a_i + \varepsilon ||a||_1) = 1 - \delta \tag{5}$$

Where $\hat{a}_i$ is the estimated value and a_i is the true value. It can be seen that the smaller ε is and the larger w is, the smaller error of the point query is; the larger δ is and the smaller d is, the smaller probability that the point query has an error is.

During the creation of Count-Min Sketch, the higher layer Count-Min Sketch has a smaller size, CBFSketch uses the smaller w and d to reduce the size of higher layer Count-Min Sketch. According to equation (1), the higher layer Count-Min Sketch is, the more impact it has on the estimated value. Therefore, the higher layer Count-Min Sketch needs to be more accurate.

IV. Implementation

A. Test Bed and Data

CBFSketch is installed in a Linux server which is configured with an Intel Xeon 2.10GHz CPU, 16GB of RAM and Broadcom Limited NetXtreme BCM5720 Gigabit Ethernet PCIe.

The data set used in the experiment is the data stream on CAIDA [22] which contains anonymous passive traffic tracking from high speed monitors on various CAIDA high speed Internet backbone links. The 2008-2014 data includes CAIDA's equinik-chicago and equinix-sanjose monitors for anonymous passive traffic captured on high-speed Internet backbone links. The 2015 to 2016 data includes anonymous passive traffic tracking from CAIDA's equinix-chicago monitor. Specifically, the data flow in the experiment is the data flow of January 21, 2016. The information of data stream is shown in TABLE I.

TABLE I
THE INFORMATION OF DATA STREAM WE USE.

Total data stream(G)	IPv4(G)	IPv6(G)	TCP(G)	UDP(G)
11.10	10.21	0.89	9.31	0.86

B. Methodology

Several typical sketches are tested in the experiment: Count-Min Sketch (CMS), CU Sketch (CUS), Count Sketch (CS), and CBFSketch (CBFS). Three aspects are compared: accuracy, memory space usage and the throughput of inserting and querying data streams.

In the experiment, the frequency of the flow is stored. The source IP address is used to classify the data stream and it is calculated by hash functions. While inserting flows, the counter is increased by 1 when each packet arrives. And the performance of CBFSketch is analyzed from three aspects: memory usage, average absolute error of estimated value and average relative error. The relative error and absolute error are calculated as follows. Absolute error of a single query is in equation (6).

$$AE_i = |V - V_{est}| \tag{6}$$

Relative error of a single query is in equation (7).

$$RE_i = \frac{|V - V_{est}|}{V} \tag{7}$$

Where V is the true value of an item and V_{est} is the query value corresponding to the item, ie the estimated value.

The average relative error and the average absolute error are the average of the errors of the query values. Average absolute error is in equation (8).

$$AAE = \frac{1}{n}\sum_{i=1}^{n} AE_i \tag{8}$$

Average relative error is in equation (9).

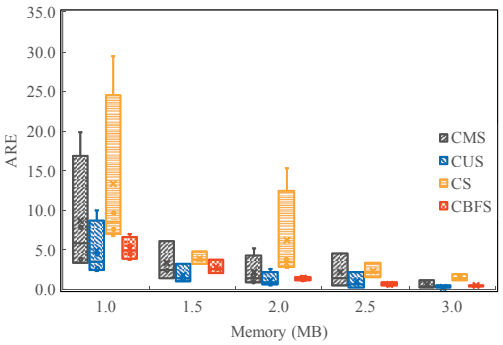
$$ARE = \frac{1}{n}\sum_{i=1}^{n} RE_i \tag{9}$$

In the experiment, w from 2^{15} to 2^{19} and d from 2 to 16 for Count-Min Sketch, CU Sketch and Count Sketch are set. And L_0 of the initial sketch of CBFSketch is from 2^{18} to 2^{20} and d from 2 to 4. The L_c of Counting Bloom Filter is from 30 percent to 60 percent of L_0 of the initial sketch. And r is from 0.5 or 0.9. We compared them at the same memory space. The memory space of CBFSketch changes with the flow size, we calculate it after inserting all items. Then we choose a similar value of the memory space to compare with the other sketches.

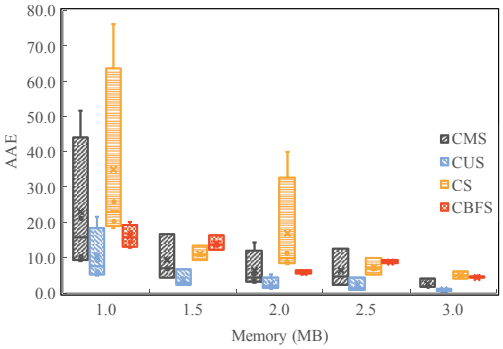
C. Accuracy

In this section, CBFSketch and several other typical sketches detect the average relative error and the average absolute error of point queries on CAIDA dataset. Fig. 5(a) and Fig. 5(b) show the average relative error and average absolute error of CBFSketch and Count-Min Sketch, CU Sketch, and Count Sketch.

As shown in Fig. 5(a), the average relative error of CBFSketch is about 40% lower than that of Count-Min Sketch when using the same memory space; the average relative error with CUS is roughly the same; it is about 45% lower than CS. CBFSketch uses the least amount of memory when the same error is reached. As shown in Fig. 5(b), the average absolute error of CBFS is higher than CUS when using the same memory space; it is basically equivalent to CMS; it is lower about 20% than CS. CBFSketch is more stable in terms of average relative error and average absolute error than several other sketches. The error of CBFSketch is stable in a relatively small range. The lower limits of error of several other sketches are similar to CBFSketch, but their upper limits are higher.



(a) Average relative errors.



(b) Average absolute errors.

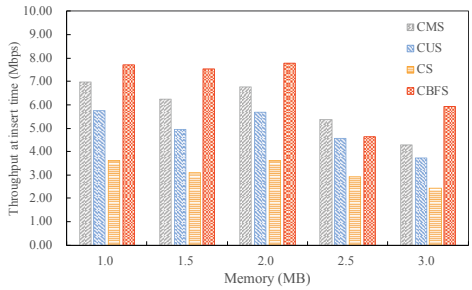
Fig. 5. Performance of accuracy.

According to CBFSketch's performance in terms of average relative error and average absolute error, CBFSketch has a significant error in using the same memory space as other sketches; and with the same precision, CBFSketch uses less memory space. The utilization of memory space is higher.

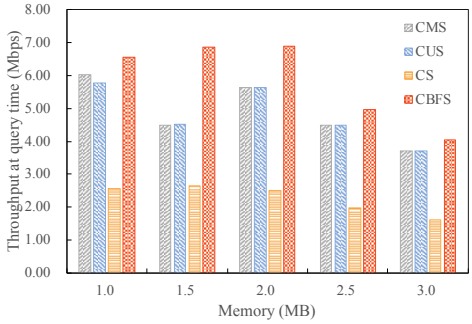
D. Throughput

In this section, Fig. 6(a) and Fig. 6(b) show the throughput of CBFSketch, Count-Min Sketch, CU Sketch, and Count Sketch.

As shown in Fig. 6(a), the throughput of inserting in CBFSketch are about 10%, 20%, and 50% higher than CMS, CUS, and CS, respectively. As shown in Fig. 6(b), the throughput of querying in CBFSketch increases about 10%, 15%, and 60%,respectively, compared to CMS, CUS, and CS. Since the sketch part of CBFSketch uses Count-Min Sketch, when compared to Count-Min Sketch, Count-Min Sketch has only one large two-dimensional array, and CBFSketch consists of several small two-dimensional arrays under the same memory space.



(a) The throughput of inserting.



(b) The throughput of querying.

Fig. 6. Performance of throughout.

V. Conclusion

Sketch is a typical hash method that has been widely used in various fields. Its main role is to aggregate and measure data streams. In this paper, we propose a framework that combines multilayer sketches with Counting Bloom Filter - CBFSketch. There is only one Count-Min Sketch and One Counting Bloom Filter initially. CBFSketch can dynamically create the next sketch if the overflow happened in the current sketch. And the size of higher layer sketch is smaller than the lower one. So there is an upper limit for the memory space of CBFSketch. In the case of maintaining a certain accuracy and throughput, the use of memory space is saved, and the utilization of memory space is improved. We compare CBFSketch with three typical sketches: Coung-Min Sketch, CU Sketch, and Count Sketch. CBFSketch performs better in terms of accuracy, throughput, and space usage. At the same time, due to the characteristics of CBFSketch, it can also be applied to abnormal traffic detection in the network security field.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 61502250, No.61702282, No. 71801123, and No. 61872191) and sponsored by NUPTSF (Grant No. NY214188). This work was funded in part by Natural Science Foundation of the Jiangsu Higher Education Institutions of China under grants 17KJB520023.

References

[1] G. He and J. C. Hou, "On sampling self-similar internet traffic," *Computer Networks*, vol. 50, no. 16, pp. 2919–2936, 2006.

[2] F. Raspall, "Efficient packet sampling for accurate traffic measurements," *Computer Networks*, vol. 56, no. 6, pp. 1667–1684, 2012.

[3] C. Estan, K. Keys, D. Moore, and G. Varghese, "Building a better netflow," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 245–256, 2004.

[4] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 153–166.

[5] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2011.

[6] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[7] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2002, pp. 693–703.

[8] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," *IEEE/ACM Transactions on Networking (TON)*, vol. 13, no. 6, pp. 1219–1232, 2005.

[9] H. Liu, Y. Sun, and M. S. Kim, "Fine-grained ddos detection scheme based on bidirectional count sketch," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*. IEEE, 2011, pp. 1–6.

[10] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "One sketch for all: fast algorithms for compressed sensing," in *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM, 2007, pp. 237–246.

[11] D. Talbot and M. Osborne, "Smoothed bloom filter language models: Tera-scale lms on the cheap," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[12] N. Polyzotis, M. Garofalakis, and Y. Ioannidis, "Approximate xml query answers," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 2004, pp. 263–274.

[13] J. Jiang, F. Fu, T. Yang, and B. Cui, "Sketchml: Accelerating distributed machine learning with data sketches," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1269–1284.

[14] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking (TON)*, vol. 8, no. 3, pp. 281–293, 2000.

[15] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *European Symposium on Algorithms*. Springer, 2002, pp. 348–360.

[16] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 2002, pp. 346–357.

[17] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory*. Springer, 2005, pp. 398–412.

[18] C. Estan and G. Varghese, *New directions in traffic measurement and accounting*. ACM, 2002, vol. 32, no. 4.

[19] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 241–252.

[20] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J.-L. Larriba-Pey, "Dynamic count filters," *Acm Sigmod Record*, vol. 35, no. 1, pp. 26–32, 2006.

[21] T. Yang, A. X. Liu, M. Shahzad, Y. Zhong, Q. Fu, Z. Li, G. Xie, and X. Li, "A shifting bloom filter framework for set queries," *Proceedings of the VLDB Endowment*, vol. 9, no. 5, pp. 408–419, 2016.

[22] [Online]. Available: http://www.caida.org/data/publications/