SF-sketch: A Fast, Accurate, and Memory Efficient Data Structure to Store Frequencies of Data Items

Tong Yang*†, Lingtong Liu‡, Yibo Yan*, Muhammad Shahzad§, Yulong Shen‡, Xiaoming Li*, Bin Cui*, Gaogang Xie¶
*Peking University, China. †Collaborative Innovation Center of High Performance Computing, NUDT, Changsha, China, 410073
‡ Xidian University, China. §North Carolina State University, USA. ¶ ICT, CAS, China.

***Abstract*—A *sketch* is a probabilistic data structure that is used to record frequencies of items in a multi-set. Sketches have been applied in a variety of fields, such as data stream processing, natural language processing, distributed data sets *etc.* In this paper, we propose a new sketch, called Slim-Fat (SF) sketch, which has a much smaller memory footprint for query while supporting updates. The key idea behind our proposed SF-sketch is to maintain two separate sketches: a small sketch called Slim-subsketch and a large sketch called Fat-subsketch. The Slim-subsketch enables fast and accurate querying. The Fat-subsketch is used to assist the insertion and deletion from Slim-subsketch. We implemented and evaluated SF-sketch along with several prior sketches and compared them side by side. Our experimental results show that SF-sketch significantly outperforms the most commonly used CM-sketch in terms of accuracy. The full version is provided at arXiv.org [12].**

I. Introduction

A *sketch* is a probabilistic data structure that is used to record frequencies of distinct items in a multi-set. Due to their small memory footprints, high accuracy, and fast speeds of queries, insertions, and deletions, several types of sketches are being extensively used in data stream processing [1], [2], [3], [4]. Sketches are also being applied in other fields, such as sparse approximation in compressed sensing [5], network anomaly detection [6], and natural language processing [7], and more [14], [15]. This paper focuses on the design of a new sketch that not only has a much smaller memory footprint compared to the existing sketches, but is also more accurate while achieving the same query speed as the best prior sketch.

Charikar *et al.* proposed the Count sketch (C-sketch) [8]. C-sketch experiences two types of errors: over-estimation error, where the result of a query is a value larger than the true value, and under-estimation error, where the result of a query is a value smaller than the true value. Improving on the C-sketch, Cormode and Muthukrishnan proposed the Count-min (CM) sketch [9], which does not suffer from the under-estimation error, but only from the over-estimation error. In a further enhancement, Cormode *et al.* proposed the conservative update (CU) sketch [10], which improves the accuracy at the cost of not supporting item deletions, *i.e.*, once the information about an item is inserted into the CU-sketch, it cannot be removed from the sketch without affecting the information of the other items in the sketch. CML-sketch [11] further improves the accuracy at the cost of suffering both over-estimation and under-estimation errors. As CM-sketch supports deletions and does not have under-estimation error, it is still the most popular sketch. However, we find that CM sketch is not compressed. The design goal of this paper is to save the space and improve the accuracy of CM-sketch while keeping its advantages.

Corresponding author: Yulong Shen {ylshen@mail.xidian.edu.cn}

In this paper, we present a new sketch, called the Slim-Fat (SF) sketch, which achieves significantly higher accuracy compared to prior art while supporting deletions and achieving the same query speed as the widely used CM-sketch. The key idea behind our proposed SF-sketch is to maintain two separate sketches: one is for query called Slim-subsketch and another is for update called Fat-subsketch. Slim-subsketch has significantly fewer counters compared to the Fat-subsketch. The motivation behind the two hierarchy architecture is to separate query and update. For example, in a distributing system for stream processing, update can be done locally on every machine, and the result should be sent to other parts of the system. Therefore, we store the a big sketch locally with all information and send a small Sketch with only the information for query in order to reduce communication overhead. In designing our SF-sketch, we start with a bare bones version of the sketch and make improvements step by step to arrive at its final design.

We use the SF_1-sketch to present the slim-fat architecture. We introduce the SF_2-sketch, which maintains an additional s-ketch called Deletion-subsketch, to support deletion. To reduce the memory usage, the SF_3-sketch gets rid of the Deletion-subsketch by making the hash functions of the slim sketch and the fat sketch corresponding. Furthermore, for the SF_4-sketch, we take advantage of locality so that the number of memory access to the Fat-sketch is minimal. Finally, we reduce over-estimation error by giving the slim sketch a smaller upper bound, and we reach the final version of the SF-sketch.

Key Contributions: 1) We propose a new sketch, namely the SF-sketch, which has higher accuracy compared to the prior art while supporting deletions and keeping the query speed unchanged. 2) We implemented C-sketch, CM-sketch, CU-sketch, CML-sketch and SF-sketch on GPU platforms to evaluate and compare the performance of all these sketches. Experimental results show that SF-sketch outperforms CM-sketch by up to 33.1 times in terms of average relative error.

II. The Slim-Fat Sketch

Next, we will present five different versions of SF-sketch, which we name SF_1-sketch through SF_4-sketch, and finally SF_F-sketch, which is our final design. Each version addresses the limitations of its predecessor version.

In our slim-fat architecture (shown in Figure 1), there is a set of arrays with fewer counters per array called a

Slim-subsketch, and a set of arrays with comparatively more counters per array called a Fat-subsketch. When inserting or deleting an item, we first update the Fat-subsketch, and then update the Slim-subsketch based on the observations we make from the Fat-subsketch. The key insight at work behind our proposed scheme is that, when inserting an item, if the value of any counter in the Slim-subsketch to which the incoming item maps is already greater than the number of times that item has already been inserted, then incrementing that counter only degrades the accuracy during the query operation. Therefore, such a counter should not be incremented. The Fat-subsketch enables us to determine whether such a counter in the Slim-subsketch is already greater than the number of times the item has already appeared.

A. SF_1: Optimizing Accuracy Using One DRAM Subsketch

As shown in Figure 1, SF_1-sketch consists of d arrays in both Slim-subsketch and Fat-subsketch. *The Fat-subsketch is exactly a standard CM-sketch with many more counters than the Slim-subsketch.* We represent the i^{th} array in the Slim-subsketch with A_i and in the Fat-subsketch with B_i. Each array in the Slim-subsketch consists of w buckets while each array in the Fat-subsketch consists of w' buckets, where $w' > w$. Furthermore, each bucket in both Slim and Fat-subsketches contains one counter. We represent the counter in the j^{th} bucket of the i^{th} array in the Slim-subsketch with $A_i[j]$, where $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant w$. Similarly, we represent the counter in the k^{th} bucket of the i^{th} array in the Fat-subsketch with $B_i[k]$, where $1 \leqslant i \leqslant d$ and $1 \leqslant k \leqslant w'$. Each array A_i is associated with a uniformly distributed independent hash function $h_i(.)$, where the output of $h_i(.)$ lies in the range $[1, w]$. Similarly, each array B_i is associated with a uniformly distributed independent hash function $g_i(.)$, where the output of $g_i(.)$ lies in the range $[1, w']$. The structure of the SF_1-sketch is shown in Figure 1.

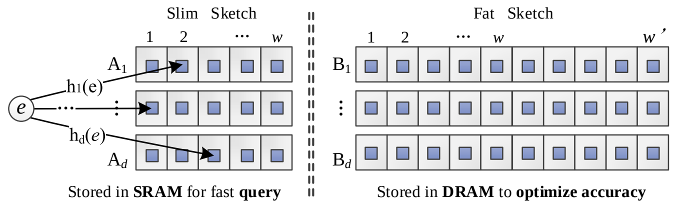
Slim Sketch
1
2
···
w
A1
h1(e)
e
···
hd(e)
Ad
Stored in SRAM for fast query
Fat Sketch
1
2
···
w
w'
B1
Bd
Stored in DRAM to optimize accuracy

Fig. 1. The Slim-Fat sketch architecture

Insertion: When inserting an item, the SF_1-sketch first inserts it into the Fat-subsketch, and based on the observations made from the Fat-subsketch, increments appropriate counters in the Slim-subsketch. To insert an item e into the Fat-subsketch, we first compute the d hash functions $g_1(e), g_2(e), \ldots, g_d(e)$ and increment the d counters $B_1[g_1(e)], B_2[g_2(e)], \ldots, B_d[g_d(e)]$ by 1. Then, we estimate its current frequency of e by finding the minimum value among all counters we just incremented and represent it with $B_e^{\min}$. To insert the item e into the Slim-subsketch, we compute the d hash functions and identify the smallest counter(s) among the d counters $A_1[h_1(e)], A_2[h_2(e)], \ldots, A_d[h_d(e)]$. If the smallest counter(s) are not smaller than $B_e^{\min}$, insertion operation ends. Otherwise, we increment the smallest counter(s) by 1. Note that if $\min_{i=1}^{d} A_i[h_i(e)] \geqslant B_e^{\min}$, we do nothing.

Query: When querying the frequency of item e, the SF_1-sketch computes the d hash functions $h_1(e), h_2(e), \ldots, h_d(e)$, and returns the value of the smallest counter among $A_1[h_1(e)], A_2[h_2(e)], \ldots, A_d[h_d(e)]$ as the result of the query. Note that the query is entirely answered from the Slim-subsketch. The query method remains the same in the latter versions, so we do not repeat it.

Advantages and Limitations: The key advantage of the SF_1-sketch is that it maintains a very small Slim-subsketch for very accurate and fast query. Unfortunately, the SF_1-sketch does not support deletions from the Slim-subsketch. We address this problem in the next version of SF-sketch.

B. SF_2: Supporting Deletion Using the Deletion-subsketch

To support deletions, in addition to the Slim- and Fat-subsketch, the SF_2-sketch maintains another sketch called the Deletion-subsketch, which is represented by C. Unlike Fat-subsketch, all the parameters (d, w, $h_i(.)$) of the Deletion-subsketch are same with those of the Slim-subsketch. The Deletion-subsketch helps in deciding which counters to decrement in the Slim-subsketch when deleting an item.

Insertion: Apart from updating Slim- and Fat-subsketches, the SF_2-sketch also insert the item e into the Deletion-subsketch by incrementing the d counters $C_1[h_1(e)], C_2[h_2(e)], \ldots, C_d[h_d(e)]$ by 1.

Deletion: To delete an item e from the SF_2-sketch, we first delete it from the Fat-subsketch by decrementing the d counters $B_1[g_1(e)], B_2[g_2(e)], \ldots, B_d[g_d(e)]$ by 1 and then delete it from the Deletion-subsketch by decrementing the d counters $C_1[h_1(e)], C_2[h_2(e)], \ldots, C_d[h_d(e)]$ by 1. Finally, we delete it from the Slim-subsketch. We leverage the fact that each counter in the Deletion-subsketch is always less than or equal to the real value of the item. Because when inserting an item, even if a counter in the Slim-subsketch to which the incoming item maps to is not incremented, the corresponding counter in the Deletion-subsketch is always incremented. As a result, the Deletion-subsketch draws an upper bound of the Slim-subsketch. To delete the item e from the Slim-subsketch, for each $i \in [1, d]$, we compare $A_i[h_i(e)]$ with $C_i[h_i(e)]$ and decrement $A_i[h_i(e)]$ by 1 when $A_i[h_i(e)] > C_i[h_i(e)]$.

Advantages and Limitations: The SF_2-sketch is advantageous over the SF_1-sketch because it supports deletions. However, it is not efficient in terms of memory usage and update speed because it has to maintain an additional sketch, the Deletion-subsketch. We address this problem in SF_3-sketch.

C. SF_3: Supporting Deletion Using One DRAM Subsketch

In SF_3-sketch, we get rid of the separate Deletion-subsketch, and modify the Fat-subsketch so that, in addition to insertions, it can assist deletions in the Slim-subsketch. In the Fat-subsketch of SF_3-sketch, the number of buckets in each array is given by $w' = z \times w$, where z is a positive integer. In other words, the Fat-subsketch consumes z times as much memory as the Slim-subsketch. In addition, the hash functions of the Slim-subsketch, $h_i(.)$, are derived from the hash functions of the Fat-subsketch, $g_i(.)$, by $h_i(.) = \Big(g_i(.) - 1\Big)\% w + 1$ Note that calculating the hash function $h_i(.)$ from the hash function $g_i(.)$ using the equation above essentially *associates* each counter $A_i[j]$ in the Slim-subsketch with z counters $B_i[j], B_i[j+w], B_i[j+2w], \ldots, B_i[j+(z-1)w]$ in the Fat-subsketch. Every time a counter in the Slim-subsketch is incremented, it is certain that one of its *associated* z counters in the Fat-subsketch is also incremented. This further means that the value of a counter in the Slim-subsketch will always be less than or equal to the sum of values of all its associated counters in the Fat-subsketch.

Insertion: When inserting an item e, the SF$_3$-sketch using the same rule with SF$_1$- and SF$_2$-sketches, except that the hash functions of the Slim-sketch is changed.

Deletion: To delete an item e from the SF$_3$-sketch, we first delete it from the Fat-subsketch by decrementing the d counters $B_1[g_1(e)], B_2[g_2(e)], \ldots, B_d[g_d(e)]$ by 1. Then we delete e from the Slim-subsketch. For each $i \in [1,d]$, we compare $A_i[h_i(e)]$ with $\sum_{m=0}^{z-1} B_i[h_i(e)+(m\times w)]$ and decrement $A_i[h_i(e)]$ by 1 if $A_i[h_i(e)] > \sum_{m=0}^{z-1} B_i[h_i(e)+(m\times w)]$. Here we leverage the fact that *the value of a counter in the Slim-subsketch should be less than or equal to the sum of values of all its associated counters in the Fat-subsketch.*

Advantages and Limitations: The advantage of SF$_3$-sketch over the SF$_2$-sketch is that it does not have to maintain a separate Deletion-subsketch. Unfortunately, it is not efficient in terms of deletion speed. Since to check the sum, it needs $d \times z$ memory accesses to add up the counters in each array of the Fat-subsketch. In the next version of our SF-sketch, we address this limitation while keeping the advantages of all three previous versions of the SF-sketch.

D. SF$_4$: Improving Deletion Speed

As shown in Figure 2, in the Fat-subsketch of the SF$_4$-sketch, we have d arrays with $w' = w$ buckets each, and each bucket now contains z counters instead of one counter. We represent the k^{th} counter in the j^{th} bucket of the i^{th} array in the Fat-subsketch with $B_i[j][k]$. Each array B_i in the Fat-subsketch is associated with two uniformly distributed independent hash functions: $h_i(.)$ mapping an item to a bucket in the i^{th} array, and $f_i(.)$ mapping an item to a counter inside the bucket $B_i[h_i(.)]$. The Slim-subsketch uses the same hash functions $h_i(.)$ as the Fat-subsketch to map items to buckets.

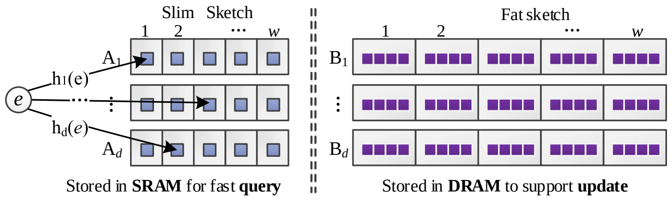
Slim Sketch
1
2
...
w
A1
Ad
h1(e)
hd(e)
e
Fat sketch
B1
Bd
Stored in SRAM for fast query
Stored in DRAM to support update

Fig. 2. The SF$_4$- and & SF$_F$-sketch architecture

Insertion: To insert an item e into the Fat-subsketch, we first compute d hash functions $h_1(e), h_2(e), \ldots, h_d(e)$ and another d hash functions $f_1(e), f_2(e), \ldots, f_d(e)$ and increment the d counters $B_1[h_1(e)][f_1(e)]$, $B_2[h_2(e)][f_2(e)]$, ..., $B_d[h_d(e)][f_d(e)]$ by 1. Next, we find the minimum value among all counters we just incremented and represent it with $B_e^{\min}$. To insert the item e into the Slim-subsketch, we identify the counters with the smallest value among the d counters $A_1[h_1(e)], A_2[h_2(e)], \ldots, A_d[h_d(e)]$ and increment them by 1 only if their values are less than $B_e^{\min}$.

Deletion: To delete the item e from the Fat-subsketch, we decrement the d counters $B_1[h_1(e)][f_1(e)]$, $B_2[h_2(e)][f_2(e)]$, ..., $B_d[h_d(e)][f_d(e)]$ by 1. To delete the item e from the Slim-subsketch, for each $i \in [1,d]$, we compare $A_i[h_i(e)]$ with $\sum_{k=1}^{z} B_i[h_i(e)][k]$ and decrement counter $A_i[h_i(e)]$ by 1 if $A_i[h_i(e)] > \sum_{k=1}^{z} B_i[h_i(e)][k]$. we leverage the fact stated earlier that *the value of a counter in the Slim-subsketch should always be less than or equal to the sum of values of all counters in the corresponding bucket in the Fat-subsketch.*

Advantages and Limitations: The advantage SF$_4$-sketch has over SF$_3$-sketch is that all counters in the Fat-subsketch corresponding to a counter in the Slim-subsketch are now located in the same bucket. Thus, one deletion from the Fat-subsketch only needs $d \times z \times b/W$ memory accesses, where b is the number of bits of each counter, W is the size of the `machine word`, and $b < W$. Based on SF$_4$-sketch, our final version SF$_F$-sketch aims to minimize the over-estimation error.

E. SF$_F$: Reducing Over-Estimation Error

The structure of the SF$_F$-sketch as well as the query and insertion process are exactly the same as those of the SF$_4$-sketch. The key idea behind the SF$_F$-sketch is that when performing a deletion, we keep the value of each counter in the Slim-subsketch always less than or equal to the value of the largest counter instead of the sum of the corresponding bucket of the Fat-subsketch.

Deletion: To delete an item e from the Slim-subsketch, we first check the d buckets $B_1[h_1(e)], B_2[h_2(e)] \ldots B_d[h_d(e)]$. For each $i \in [1,d]$, if $\max_{k=1}^{z} B_i[h_i(e)][k]$ changes when deleting item e from the Fat-subsketch, we set $A_i[h_i(e)] = \max_{k=1}^{z} B_i[h_i(e)][k]$ if $A_i[h_i(e)] > \max_{k=1}^{z} B_i[h_i(e)][k]$. Otherwise, we leave the value of $A_i[h_i(e)]$ unchanged.

Advantages: The key advantage of SF$_F$-sketch over SF$_4$-sketch is that during deletion operation, it significantly reduces the counter values in the Slim-subsketch, and furthermore reduces the over-estimation error of SF$_F$-sketch. We should mention that SF$_F$-sketch does not suffer from under-estimation error, which is formally proved in the full version of this paper [12].

III. Experimental Results

A. Experimental Setup

We present the experimental results on skewed dataset in this paper, which are generated by the well known YCSB [13]. We keep the skewness of our skewed dataset equal to the `default value` for YCSB, which is 0.99. Experiments on more kinds of datasets can be seen in the full version of our paper [12].

B. Experiments on Accuracy

We use *relative error* (RE) to quantify the accuracy of sketches. Let f_e represent the actual frequency of an item e and let $\hat{f}_e$ represent the estimate of the frequency returned by the sketch, the relative error is defined as the ratio $|\hat{f}_e - f_e|/f_e$. To evaluate accuracy, we used 100K distinct items and fixed parameter setting (d = 5, w = 40000, z = 3). We calculated

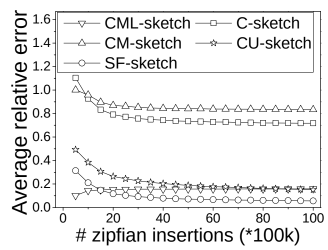

Fig. 3. Average relative error vs. number of insertions (skewed)

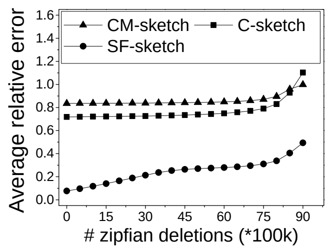

Fig. 4. Average relative error vs. number of deletions (skewed)

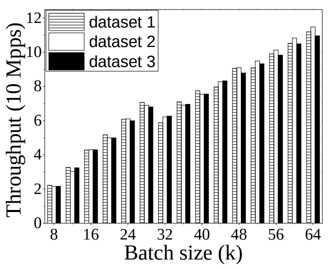

Fig. 5. Throughput vs. batch size

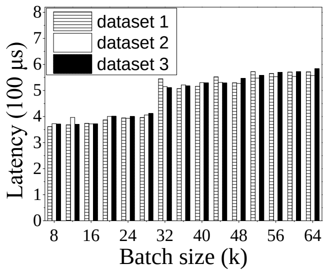

Fig. 6. latency vs. batch size

relative errors for different sketches in two settings: (1) by incrementally increasing the number of insertion operations; (2) by incrementally increasing the number of deletion operations.

Relative Error vs. # of Insertions: *Our experimental results, reported in Figure 3, show that in case of skewed workload, the average relative error of SF-sketch is* $[0.3, 2.8]$, $[3.2, 12.7]$, $[3.0, 14.8]$, *and* $[1.5, 2.7]$ *times smaller than the average relative errors of CML, C, CM, and CU-sketches, respectively.* The converged average relative error of our SF-sketch is 2.8, 12.7, 14.8 and 2.7 times smaller than the converged average relative errors of CML, C, CM, and CU-sketch, respectively.

Relative Error vs. # of Deletions: *Our experimental results, reported in Figure 4, show that for skewed workload, the average relative error of SF-sketch is [2.1 to 12.7] and [1.9 to 14.8] times smaller than the average relative errors of C and CM-sketches, respectively.*

C. Experiments on query speed

We measure two metrics in our experiments on GPU: throughput and average query latency. We did experiments on a range of CUDA configurations: stream ($1 \sim 24$), block ($64 \sim 512, step = 32$), and thread ($128 \sim 1024, step = 128$). We observed that the throughput of all five sketches (*i.e.*, CML, C, CM, CU and SF-sketches) was almost the same. For this reason, we only present results for SF-sketch.

Our experimental results for three different data sets show that the query speed in GPU increases with the increase in the batch size. As shown in Figure 5, for the batch size of 20K queries, the query speed is around 50 million queries per second (Mqps). With increase in the batch size, SF-sketch reaches a query speed higher than 110 Mqps.

Our experimental results for three different data sets show that for SF-sketch, to reduce latency, the batch size of 28K is the most optimal for our experimental setup. Figure 6 shows that the the average query latency of SF-sketch is below $410\ \mu s$ for batch sizes $\leqslant 28K$. For batch sizes $\geqslant 32k$, the latency increases to $511 \sim 584\ \mu s$.

IV. Conclusion

In this paper, we proposed a new sketch, namely the SF-sketch, which achieves up to 14.8 times higher accuracy compared to the CM-sketch while using the same amount of memory for query and keeping the query and update speeds as fast as the CM-sketch. The key idea behind our proposed SF-sketch is to maintain two separate sketches, one for query called Slim-subsketch and another for supporting update called Fat-subsketch. The Slim-subsketch enables SF-sketch to achieve high query speed and small communication overhead while the Fat-subsketch enables it to achieve high query accuracy. To evaluate and compare the performance of our proposed SF-sketch, we conducted experiments on GPU platforms. Our experimental results show that our SF-sketch significantly outperforms the-state-of-the-art in terms of accuracy.

Acknowledgment

This work is partially supported by Primary Research & Development Plan of China (2016YFB1000304), National Basic Research Program of China (2014CB340400), NSFC (61232004, 61672061), and CAS-NDST Lab, ICT, CAS, 100190, Beijing, China, Special Fund for strategic pilot technology Chinese Academy of Sciences (XDA06040602), and National Science Foundation, USA (CNS-1616317).

References

[1] C. C. Aggarwal and S. Y. Philip. On classification of high-cardinality data streams. In *SDM*, volume 10, pages 802–813. SIAM, 2010.

[2] A. Chen, Y. Jin, J. Cao, and L. E. Li. Tracking long duration flows in network traffic. In *Proc. IEEE INFOCOM*, 2010.

[3] G. Cormode and M. Garofalakis. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large data bases*, pages 13–24. VLDB Endowment, 2005.

[4] T. Yang, A. Liu, and M. Shahzad, et al. A Shifting Bloom Filter Framework for Set Queries. Proceedings of the VLDB Endowment. 2016:408-419.

[5] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of ACM symposium on Theory of computing*, 2007.

[6] C. Callegari and N. Cyprus. Statistical approaches for network anomaly detection. In *Proc. ICIMP*, 2009.

[7] B. Van Durme and A. Lall. Probabilistic counting with randomized storage. In *IJCAI*, pages 1574–1579, 2009.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*. Springer, 2002.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[10] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *ACM SIGCOMM CCR*, 32(4), 2002.

[11] G. Pitel and G. Fouquier. Count-min-log sketch: Approximately counting with approximate counters. *arXiv :1502.04885*, 2015.

[12] T. Yang, L. Liu, Y. Yan and et al. SF-sketch: A Two-stage Sketch for Data Streams. *arXiv, paper id: 1701.04148 [cs.DS]*, 2017.

[13] B. F. Cooper, A. Silberstein, and et al. Benchmarking cloud serving systems with YCSB. In *Proc. ACM SOCC*, 2010.

[14] T. Yang, G. Xie, Y. Li, Q. Fu, et al. Guarantee IP lookup performance with FIB explosion. ACM SIGCOMM, 2014.

[15] Dharmapurikar, Sarang and Krishnamurthy, Praveen and Taylor, David E. Longest prefix matching using bloom filters. ACM SIGCOMM, 2003.