IEEE TRANSACTIONS ON
PARALLEL AND
DISTRIBUTED SYSTEMS

SF-sketch: A Two-stage Sketch for Data Streams

Lingtong Liu, Yulong Shen, Yibo Yan, Tong Yang, Muhammad Shahzad, Bin Cui, Gaogang Xie

Abstract—Sketches are probabilistic data structures designed for recording frequencies of items in a multi-set. They are widely used in various fields, especially for gathering Internet statistics from distributed data streams in network measurements. In a distributed streaming application with high data rates, a sketch in each monitoring node "fills up" very quickly and then its content is transferred to a remote collector responsible for answering queries. Thus, the size of the contents transferred must be kept as small as possible while meeting the desired accuracy requirement. To obtain significantly higher accuracy while keeping the same update and query speed as the best prior sketches, in this paper, we propose a new sketch – the Slim-Fat (SF) sketch. The key idea behind the SF-sketch is to maintain two separate sketches: a larger sketch, the Fat-subsketch, and a smaller sketch, the Slim-subsketch. The Fat-subsketch is used for updating and periodically producing the Slim-subsketch, which is then transferred to the remote collector for answering queries quickly and accurately. We also present the error bound as well as an accurate model of the *correct rate* of the SF-sketch, and verify their correctness through experiments. We implemented and extensively evaluated the SF-sketch along with several prior sketches. Our results show that when the size of our Slim-subsketch and of the widely used Count-Min (CM) sketch are kept the same, our SF-sketch outperforms the CM-sketch by up to 33.1 times in terms of accuracy (when the ratio of the sizes of the Fat-subsketch and the Slim-subsketch is 16:1). We have made all source codes publicly available at Github [1].

Index Terms—Network measurements, sketch, distributed monitoring, multiset, frequent items.

1 Introduction

1.1 Background and Motivation

SKETCHES are probabilistic data structures designed for recording frequencies of distinct items in a multi-set. Due to their small memory footprints, high accuracy, and fast speeds of queries, insertions, and deletions, sketches are extensively used in data stream processing [3], [4], [5], [6], [7], [8]. Especially in the distributed data stream monitoring scenario [9], [10], [11], as the network statistics used for query and analysis in the remote control center are aggregated using local sketches maintained by the distributed nodes, not only the query accuracy must meet the minimum constraints, but also the size of local information transmitted in the network to the center must be kept as small as possible so as not to harm the regular data streams' transmission in it.

Towards measuring the size or frequencies of each data stream in local nodes, most works [12], [13], [14], [15] focus on how to improve accuracy for query and analysis in most of these distributed data stream applications, such as sensor database, network traffic, graph streams, web streams and multi-media streams. Unfortunately, the impact of the size of local sketches on the link capacity, when transmitted in the network, is overlooked. As an effect, the sketch "fills up" very quickly, especially when deployed in high speed streaming scenarios. When transferring the local filled-up sketch to some remote collector, the transmission consumes some portion of the link bandwidth. Thus, it is quite important to keep the sizes of the sketches in each node as small as possible so that the bandwidth-heavy and speed-critical data streams can transfer uninterrupted. This paper focuses on the design of a new scheme that significantly reduces the size of the sketch transferred compared with the existing sketches, and is also more accurate while maintaining the same query and update speed as the best prior sketches.

1.2 Limitations of Prior Art

A data stream can be thought of as a sequence of m discrete tuples (e_i, f_i) over a domain $E \times F$, in which $E = \{1, 2, \ldots, n\}, F \subset R,$ and $i \in \{1, 2, \ldots, m\}$. Monitored data stream in various scenarios can be divided into two different types according to F. The *Cash register model* is used to model the situation where the item e_i being monitored arrives in some arbitrary order with a positive value f_i, whereas the *Turnstile model* models the situation where the value of f_i can be any real number. Examples for *Cash register* model can be found in counting the number of packets or estimating the size of each item transmitted in a network. Bank depositors' saving and withdrawal can be interpreted as a financial *Turnstile* data stream, a synopsis of which can facilitate real-time analysis and decision making.

Three typical sketches include the Count (C) sketch, the Count-Min (CM) sketch, and the Conservative Update (CU) sketch. Charikar *et al.* proposed the C-sketch [16]. As the frequency of an item is estimated by taking the median of various counters, it experiences two types of errors: overestimation, where the estimated query result is larger than the real value, and underestimation, where the estimated query result is smaller. Cormode and Muthukrishnan proposed the CM-sketch [17], which uses the minimum of various counters as an estimate of the frequency of a given item. The

- *L. Liu and Y. Shen are with the Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an 710071, China and the School of Computer Science and Technology, Xidian University, Xi'an 710071, China. E-mail: xviviliu@gmail.com, ylshen@mail.xidian.edu.cn.*
- *T. Yang, Y. Yan and B. Cui are with the Department of Computer and Science, Peking University, Beijing 100871, China. E-mail: yangtongemail@gmail.com, {yanyibo, bin.cui}@pku.edu.cn.*
- *M. Shahzad is with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695 USA. E-mail: mshahza@ncsu.edu*
- *G. Xie is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China. E-mail: xie@ict.ac.cn*

This paper is an extension to a poster [2].

Digital Object Identifier 10.1109/TPDS.2020.2987609

CM-sketch suffers only from overestimation. The authors claimed that such one-sided error has many benefits, such as an approximate query for heavy hitters in a data stream with a sketch that never underestimates finds every heavy hitter. Note that both sketches above are linear sketches and can support *Turnstile* and *Cash register* models because the sketch summary is a linear transformation of the input stream. The CU-sketch [18], proposed by Cormode *et al.*, improves the query accuracy while sacrificing linearity, which makes it incapable to support deletions, and only works for *Cash register* model. Currently, the CM-sketch is the most popular in practice because it supports deletions and has no underestimation error. As mentioned above, given the desired accuracy and transmission period, a smaller sketch leads to lesser requirements for bandwidth and costly fast memory, and to faster query speed in the collector.

1.3 Proposed Approach

In this paper, we present a novel two-stage sketch named the Slim-Fat (SF) sketch. To better understand the key idea of our SF-sketch, let us first look at the update procedure of the CU-sketch: by using the auxiliary information of the minimum of various counters for the incoming item as a dynamic bar, only counters not bigger than the bar are updated. Inspired by the conservative updating of the CU-sketch, the SF-sketch updates the summary in a more conservative way. To update the SF-sketch for an incoming item, a subset of counters chosen from some *right counter set* is used to generate the dynamic bar. Out of the other subset of counters chosen from some *left counter set*, only the counters that are not bigger than the auxiliary bar are updated. Specifically, the SF-sketch maintains two separate sketches (as shown in Figure 2): a large sketch called the Fat-subsketch, composed of the *right counter set* for generating dynamic bar, and a small sketch called the Slim-subsketch composed of the *left counter set*. Given an incoming item, the Fat-subsketch is first modified by updating the associated *right counters*, from which auxiliary information is generated. This information is then used to guide the updating of the Slim-subsketch. Further query operations are only applied to the Slim-subsketch. Both subsketches have similar accuracy (see Section 4), and thus, we only need to send the Slim-subsketch to the remote collector. Compared to the state-of-the-art, the Slim-subsketch achieves significantly higher accuracy and the same query speed when using the same size of memory.

The Fat-subsketch is, in fact, a CM-sketch physically. As shown in Figure 1, a CM-sketch consists of d arrays $\{\mathbf{A}_i, 1 \leq i \leq d\}$. Each array consists of w counters. We represent the counter in the j^{th} position of the i^{th} array as $A_i[j]$. Each array $\mathbf{A}_i$ is associated with an independent hash function $h_i(.)$ with range $[1, w]$. The initialization for the CM-sketch is simply to set all the $d \times w$ counters to zero. Given an item e, d *right counters* are chosen from those d arrays each at a position calculated by one of the d hash functions. To update a tuple (e, f), each of the d *right counters* is updated by f. When querying the frequency of e, it returns the minimum counter among the d *right counters*.

The Slim-subsketch is, in fact, a conservative CU-sketch logically, and as the name suggests, has significantly fewer counters compared to the Fat-subsketch. The number of

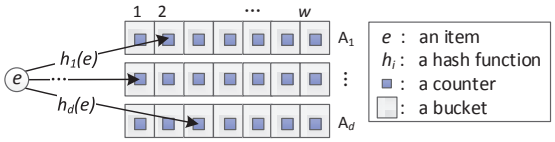


Fig. 1: The Count-Min sketch architecture.

arrays and hash functions is the same as the Fat-subsketch. Each counter in the i^{th} array is linked to a set of counters in the corresponding i^{th} array of the Fat-subsketch. As all the items that hashed to some specific counter of the Slim-subsketch array also hashed to some set of counters of the Fat-subsketch array, the left counter is linked to the right set of counters. As a result, the Slim-subsketch is logically linked to the Fat-subsketch. Given an input item, this link gives us a way to apply some conservative updates to the left counters by using information from the right counters.

Different update strategies lead to different versions of SF-sketch. Details of those strategies will be discussed in Section 3. Here, we give an overview of the three SF-sketches. Two intermediate versions are interactive, while the final version is two-stage. By $interactive$, we mean that for each item's arrival, the update algorithm has to be applied in both the Fat-subsketch and the Slim-subsketch cooperatively. The *two-stage* version updates only the Fat-subsketch for any arbitrary items' arrival, and then transforms the fat one into slim quickly once needed, such as when transferring its content to a remote collector. Note that the two intermediate versions can be viewed as two variants that are of independent interests and can be applied in different scenarios. Based on the slim-fat idea, to make every memory space count, we first propose the SF_1-sketch in a cash-register model, which can't support deletions. To support deletions for stream in strict *turnstile model*, we propose the SF_2-sketch in which the size of the Fat-subsketch must be an integer multiple of the slim one. To boost updating speed, we propose the final two-stage SF_F-sketch that only updates the Fat-subsketch for each item's arrival. It then compresses the Fat-subsketch in a simple and fast way using SIMD [19] to generate the Slim-subsketch.

1.4 Key Contributions

The previous version of this paper was presented in ICDE 2017 [2]. This journal version significantly extends the previous version by adding many technical details, mathematical proofs, and experimental results.

1) We propose a new sketch, namely the SF-sketch, which has higher accuracy compared to the prior art while supporting deletions and keeping comparable update and query speeds.
2) We theoretically analyzed several key aspects of the proposed SF-sketch, and experimentally verified the correctness of the derived "correct rate".
3) We implemented the C-sketch, CM-sketch, A-sketch, CU-sketch and SF-sketch on GPU and multi-core CPU platforms and carried out extensive experiments on these two platforms to evaluate and compare the performance of all these sketches. Our results on real and synthetic datasets show that the SF-sketch outperforms the CM-sketch by up to 33.1 times in terms of average

relative error when configuring the ratio of the size of the Fat-subsketch and the Slim-subsketch to 16:1.

2 Related Work

The C-sketch, the CM-sketch, and the CU-sketch are three fundamental frequency based sketches used to maintain the summary of the monitored data streams. As the C-sketch [16] suffers from both overestimation and underestimation, subsequently proposed improvements, *e.g.* the CM-sketch [17] and CU-sketch [18], suffer from overestimation only. Although the CU-sketch improves the query accuracy significantly, its fundamental limitation is that it does not support deletions, and consequently it has not received as wide acceptance in practice as the CM-sketch. Thorough statistical analyses of various sketches are provided in [20], [21].

As a universal framework, the Augment (A) sketch [7] can be applied to many existing sketches. The A-sketch first uses a filter to catch heavy hitters (high-frequency items), and then uses a classical sketch (such as a CM-sketch or a C-sketch) to store and query the remaining items. However, always keeping the most frequent items in the filter without incurring additional errors is a challenging issue. Unavoidability of complex design and frequent interaction between the two components makes the implementation complicated. For improving update speed, the authors recommended using a filter with 32 items. The authors also showed accuracy improvement by mainly querying very frequent items. Without the details of the query sets, we cannot reproduce their experimental results. To make our experimental results reproducible, we query each distinct item once, and find that when the A-sketch is applied to a CM-sketch, the average relative error stays almost unchanged.

A recent work presented the Diamond Sketch [22], which dynamically assigns to each flow a proper number of atom sketches in order to reduce memory footprint. Though it greatly reduces the relative error, it is designed specifically for flows with non-uniform distribution while our proposed SF-sketch can be applied to flows with arbitrary distribution. We have, therefore, not included the Diamond Sketch in our experiments for comparison.

Another class of data structures that can be used to store frequencies of items are the enhanced Bloom filters [23], such as Spectral Bloom Filters (SBF) [24], Dynamic Count Filters (DCF) [25], the counting quotient filter [26] and more [27]. The SBF replaces each bit in the conventional Bloom filter by a counter [24]. To insert an item, the basic version of the SBF simply increments all the counters that the item maps to. When querying the frequency of an item, the SBF returns the value of the smallest counter(s) among all the counters to which the hash functions map the item to. The SBF is not memory efficient enough, and is improved by the DCF by using two separate filters [25]. The first filter is comprised of fixed size counters while the size of counters in the second filter is dynamically adjusted. The use of two filters, unfortunately, increases the complexity of the DCF, which degrades its query and update speeds. The counting quotient filter (CQF) [26] is an enhanced version of quotient filters [28], which is designed to replace Bloom filters. The CQF is an excellent data structure because it can achieve better accuracy than the standard Bloom filters [23]. To achieve memory efficiency, the size of the CQF should be set to 2^n (n is an arbitrary integer) bits. Meanwhile, if the number of items is not known a priori, dynamically resizing to double is needed whenever its load factor is sufficiently high (*e.g.*, 95%). Consider the situation that the size of a CQF reaches a maximum memory usage budget. The CQF may not update well for incoming items if it is nearly full. The size of the CQF is heavily related to the frequency vector of items. Given the dynamically changing distributed data streams and possible decrease in the bandwidth budget available for transferring the summary of the local data stream to a remote collector, the CQF may not be a better option.

While most of the works assume that a sketch can been stored on main memory, Goswami *et al.* [29] consider the orthogonal situation that the CM-sketch may become too large to maintain in memory and should be migrated to external storage (*e.g.*, Solid State Drive). They proposed the buffered CM-sketch [29] that reduces the number of I/O accesses to the CM-sketch hosted by external storage when performing query/update operations.

3 The Slim-Fat Sketch

In this section, we present the details of our SF-sketch. To better explain the intuition at work behind the SF-sketch and to justify the design choices we made in developing the SF-sketch, we will start with a basic version and improve it incrementally to arrive at the final design. For each intermediate version of the SF-sketch that we develop while working our way towards the final design, we will first describe its insertion, query, and deletion operations. After that we will discuss its limitations, which will guide us in making our design choices for the next version. In this process, we will present three different versions of the SF-sketch, which we name the SF_1-sketch, the SF_2-sketch, and finally the SF_F-sketch, which is our final design. Each version is developed by studying the limitations of its predecessor version and addressing them.

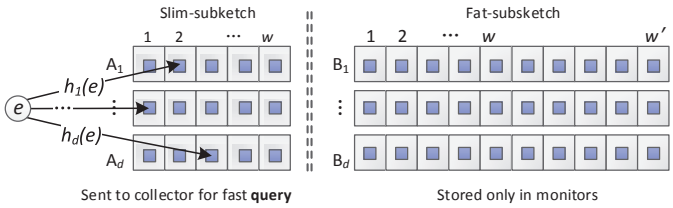


Fig. 2: The Slim-Fat sketch architecture.

Rationale: In our slim-fat architecture (shown in Figure 2), 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 smallest one of the d hashed counters is already bigger than its current true frequency, then incrementing any counter only degrades the accuracy.* As the true accuracy is not easy to obtain using small memory, we use the Fat-subsketch

to give a relatively accurate estimate of the current true frequency. Next, we start with the first version of our Slim-Fat sketch, *i.e.*, the SF_1-sketch, and discuss its operations and limitations, which will pave the way towards the design of its subsequent versions.

3.1 SF_1: Optimizing Accuracy Using a Fat-Subsketch

As shown in Figure 2, the SF_1-sketch consists of d arrays in both the Slim-subsketch and the 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 $\mathbf{A}_i$ and in the Fat-subsketch with $\mathbf{B}_i$. 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 the 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 $\mathbf{A}_i$ 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 $\mathbf{B}_i$ is associated with a uniformly distributed independent hash function $g_i(.)$, where the output of $g_i(.)$ lies in the range $[1, w']$. The **initialization** operation for the SF_1-sketch is simply setting all counters $A_i[j]$ and $B_i[k]$ to zero.

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. *The insertion operation in the Fat-subsketch is exactly the same as the conventional CM-sketch.* 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 hashed counters $B_1[g_1(e)], B_2[g_2(e)], \ldots, B_d[g_d(e)]$ by 1. For ease of explanation, we assume that the increment is 1 instead of any arbitrary positive integer. After inserting the item, we estimate the current frequency of e by finding the minimum value among the d hashed 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 hashed counters $A_1[h_1(e)], A_2[h_2(e)], \ldots, A_d[h_d(e)]$. If the value of 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 the CU-sketch always increments the smallest counter(s). Thus the SF_1-sketch is more accurate than the CU-sketch. In other words, $\forall l \in [1, d]$, the SF_1-sketch increments all counters $A_l[h_l(e)]$ by one that satisfy the following two conditions: $A_l[h_l(e)] = \min_{i=1}^{d} A_i[h_i(e)]$, and $A_l[h_l(e)] < B_e^{\min}$.

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 answered only from the Slim-subsketch.

Deletion: The SF_1-sketch does not support deletions.

Advantages and Limitations: The key advantage of the SF_1-sketch is that to answer a query it only accesses the Slim-subsketch, which keeps the query speed of this sketch as fast as the conventional CM-sketch. Furthermore, note that during the insertion operation, we either increment no counters or increment only the smallest counter(s) in the Slim-subsketch. The smallest counter in the Fat-subsketch gives the upper bound on the number of times that a given item has already been inserted. This strategy reduces the number of increments in the Slim-subsketch, which has two advantages. First, it reduces the memory footprint of the Slim-subsketch on the expensive and limited memory. Second, due to the fewer increments, the overestimation is reduced. Unfortunately, the biggest limitation of the SF_1-sketch is that it does not support deletions from the Slim-subsketch. While the Fat-subsketch assists the Slim-subsketch during insertion operation, it cannot assist in the deletion operation because the numbers of counters per array in the Fat- and Slim-subsketches are not the same. This inability to support deletions from the Slim-subsketch limits the practical usability of the SF_1-sketch. In the next version of our SF-sketch, *i.e.*, the SF_2-sketch, we address this limitation while keeping the advantages of the SF_1-sketch.

3.2 SF_2: Make Fat-Subsketch Support Deletions

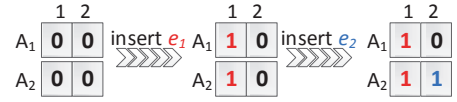


Fig. 3: An example of the deletion problem.

Difficulties for deletions: It is challenging to achieve accurate deletions in the SF_1-sketch because to delete items from the Slim-subsketch of the SF_1-sketch, one has to keep track of exactly which counters were incremented when inserting each item. Such tracking is difficult and requires large memory and processing overhead. We explain this with the help of an example. As shown in Figure 3, consider a Slim-subsketch that has two arrays and two counters per array, where all counters are initialized to 0. Let us first insert two items e_1 and e_2 and then delete the item e_1. Furthermore, let e_1 maps to $\mathbf{A}_1[1]$ and $\mathbf{A}_2[1]$ and e_2 maps to $\mathbf{A}_1[1]$ and $\mathbf{A}_2[2]$. In inserting e_1, we increment $\mathbf{A}_1[1]$ and $\mathbf{A}_2[1]$ both to 1. After that, in inserting e_2, as the current value of $\mathbf{A}_1[1]$ is 1 and $\mathbf{A}_2[2]$ is 0, we only increment the smaller of the two, *i.e.*, $\mathbf{A}_2[2]$ to 1. At this point, $\mathbf{A}_1[1] = 1$, $\mathbf{A}_1[2] = 0$, $\mathbf{A}_2[1] = 1$, and $\mathbf{A}_2[2] = 1$. In deleting e_1, as e_1 maps to both $\mathbf{A}_1[1]$ and $\mathbf{A}_2[1]$ and as both were incremented at the time of inserting e_1, if we decrease them both, the query result of e_2 will be 0, *i.e.*, an underestimation occurs, which we want to avoid in our SF-sketch.

Support Deletion: Given i and j, the counter $A_i[j]$ is linked to several counters in B_i implicitly. For the SF_1-sketch, this link can be calculated by enumerating each item e for which $h_i(e) = j$, and applying $g_i(e)$ to get corresponding set of counters S_j in B_i. Let us look at the insertion procedure of the SF_1-sketch. An increase in any counter in set S_j will affect $A_i[j]$ only, given $d = 1$. So, to support deletion, we should take into consideration all counters in S_j to decide whether or not to decrease $A_i[j]$. Assuming that we know this link a priori, it is easy to devise a deletion algorithm for $d = 1$, *i.e.* $A_i[j]$ can be set to the value of the maximum counter $B_i[k]$ in S_j if and only if $A_i[j] > B_i[k]$. This assumption is, in fact, impractical, as the number of distinct

items is generally unknown and huge. But we can make that link fixed if we redesign those two set of hash functions $h_i(.)$ and $g_i(.)$. This leads to the SF_2-sketch.

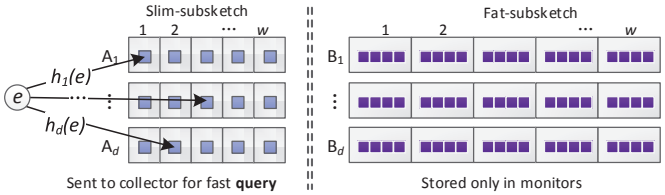


Fig. 4: The SF_2- and SF_F-sketch architecture.

As shown in Figure 4, the Fat-subsketch in the SF_2-sketch is physically the same as the Fat-subsketch in the SF_1-sketch except that the number of counters in each array of the Fat-subsketch is restricted to be an integer multiple of the counters in the Slim-subsketch. In other words, the Fat-subsketch consumes z times as much memory as the Slim-subsketch. Meanwhile, the hash functions $h_i(.)$, where $1 \leqslant i \leqslant d$, associated with the Slim-subsketch are now derived from the hash functions $g_i(.)$, where the output of $g_i(.)$ lies in the range $[1, z \times w]$. With fairness, we should link exactly z counters exclusively in the Fat-subsketch to each counter in the Slim-subsketch. While performing a deletion operation, we must find all those z counters in the Fat-subsketch to get the maximum counter. To make memory access efficient for those z counters, we put them in a continuous memory space – a bucket, so that with only one memory access, z counters can be enumerated from that bucket. More specifically,

$$h_i(.) = \lfloor (g_i(.) + z - 1)/z \rfloor \quad (1)$$

Consequently, the value of the hash function $h_i(.)$ always lies in the range $[1, w]$, where w is the number of buckets per array in the Slim-subsketch. 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 $B_i[(j-1)\times z+1], B_i[(j-1) \times z + 2], B_i[(j-1) \times z + 3], \ldots, B_i[j \times z]$ 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 value of the maximum counter of all its associated z counters in the Fat-subsketch.

Insertion & Query: The insertion and query operation of the SF_2-sketches is exactly the same as that of the SF_1-sketches, except that the set of hash functions $\{h_i(.)\}$ is defined by Equation 1.

Deletion: To delete an item from the SF_2-sketch, we first delete it from the Fat-subsketch and then from the Slim-subsketch. To delete the item e from the Fat-subsketch, we first calculate the d hash functions $g_1(e), g_2(e), \ldots, g_d(e)$ and then decrement the d counters $B_1[g_1(e)], B_2[g_2(e)], \ldots, B_d[g_d(e)]$ by 1. To delete e from the Slim-subsketch, we leverage the fact stated earlier that before deleting the item from the Fat-subsketch, *the value of a counter in the Slim-subsketch is always less than or equal to the sum of the values of all its associated counters in the Fat-subsketch*, because when inserting an item, even if a counter in the Slim-subsketch is not incremented, one of the associated counters in the Fat-subsketch is always incremented. To delete the item e from the Slim-subsketch, after deleting it from the Fat-subsketch, for each $i \in [1, d]$, we compare $A_i[h_i(e)]$ with $B_i^{max} = \max_{m=1}^{z} B_i[(h_i(e)-1) \times z + m]$ and decrease $A_i[h_i(e)]$ by 1 if and only if $A_i[h_i(e)] > B_i^{max}$.

Advantages and Limitations: Compared to the SF_1-sketch, the SF_2-sketch can be applied in scenarios with $turnstile$ data stream since it supports deletions. For each array's updating, it is relatively faster than the SF_1-sketch, as it saves one hash computation. The design of hash function $h_i(.)$ also boosts deletion operation as the z corresponding counters in the Fat-subsketch can be emulated with only one memory access. To update an item, we need $2d$ memory accesses and d hash computations to maintain the SF_2-sketch up to date, while the CM-sketch only needs d memory accesses and hash computations. Many applications that use sketches should process rapidly evolving data stream. In order to get closer to the update speed of the CM-sketch, a design improvement is needed to reduce the d memory accesses.

3.3 SF_F: The Final Version

In real data stream applications, the incoming data is continuous, and thus the sketch will "fill up" soon. Therefore, the sketch is often periodically sent to a remote collector for answering queries [30], [31], [32]. The sketch to be sent should be as small and accurate as possible. Towards this goal, we propose a new strategy including three steps: 1) performing insertion and deletion operations only on the Fat-subsketch for successive incoming items; 2) periodically producing the Slim-subsketch and sending it to the remote collector; and 3) performing query operations only on the Slim-subsketch.

Insertion & Deletion for Fat-subsketch: The insertion and deletion operations of the Fat-subsketch are exactly the same as those of the Fat-subsketch of the SF_2-sketch.

Producing Slim-subsketch: For each i, j, calculate $B_i^{max} = \max_{m=1}^{z} B_i[(i-1) \times z + m]$, and set the value of the corresponding counter $A_i[j]$ in the Slim-subsketch to B_i^{max}. In this way, we produce the Slim-subsketch for answering queries. One may wonder about the speed complexity of this process of generating a Slim-subsketch from a Fat-subsketch. Fortunately, this process can be made very fast when using SIMD (Single Instruction Multiple Data). More details on this will be provided in Section 5.2.

Query: The query operation of the Slim-subsketch is exactly the same as the previous versions of the SF-sketch.

Advantages and Limitations: The SF_F-sketch achieves faster update speed at the cost of slightly lower accuracy compared to the SF_2-sketch, because the process of producing the Slim-subsketch is equivalent to that setting the counter in the Slim-subsketch equal to the maximum value not only for deletion but also for insertion. If the frequency of requesting a new Slim-subsketch is higher than the reciprocal of the time needed for producing a Slim-subsketch, the SF_2-sketch is preferred. This is also the reason why we devise an SIMD version to speed up the Slim-subsketch's generation process. In Section 4.4, we will prove that the SF_F-sketch does not suffer from underestimation.

4 Theoretical Analysis

4.1 Bound on Overestimation

As a query is entirely answered from the Slim-subsketch, the overestimation of the SF_F-sketch is actually that of the Slim-subsketch. Therefore, next, we calculate the overestimation of the Slim-subsketch of the SF_F-sketch. Let α represent the average number of counters in any given array of the Slim-subsketch that are incremented per insertion. Note that for the standard CM-sketch, the value of α is equal to 1 because in the standard CM-sketch, exactly one counter is incremented in each array when inserting an item. For the Slim-subsketch in the SF_F-sketch, α is less than or equal to 1 because the Fat-subsketch helps in reducing the number of counters that are incremented in the Slim-subsketch per insertion. For any given item e, let $f_{(e)}$ represent its actual frequency and let $\hat{f}_{(e)}$ represent the estimate of its frequency returned by the Slim-subsketch of the SF_F-sketch. Let N represent the total number of insertions of all items into the SF_F-sketch. Let $h_i(.)$ represent the hash function associated with the i^{th} array of the Slim-subsketch, where $1 \leqslant i \leqslant d$. Let $X_{i,(e)}[j]$ be the random variable that represents the difference between the actual frequency $f_{(e)}$ of the item e and the value of the j^{th} counter in the i^{th} array, *i.e.*, $X_{i,(e)}[j] = A_i[j] - f_{(e)}$, where $j = h_i(e)$. Due to hash collisions, multiple items will be mapped by the hash function $h_i(.)$ to the counter j, which increases the value of $A_i[j]$ beyond f_e and results in overestimation. As all hash function have uniformly distributed output, $Pr[h_i(e_1) = h_i(e_2)] = 1/w$. Therefore, the expected value of any counter $A_i[j]$, where $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant w$, is $\alpha N/w$. Let ϵ and δ be two numbers that are related to d and w as follows: $d = \lceil \ln(1/\delta) \rceil$ and $w = \lceil \exp/\epsilon \rceil$. The expected value of $X_{i,(e)}[j]$ is given by the following expression.

$$\begin{aligned} E(X_{i,(e)}[j]) &= E(A_i[j] - f_{(e)}) \\ &\leqslant E(A_i[j]) \\ &= \frac{\alpha N}{w} \leqslant \frac{\epsilon \alpha}{\exp} N \end{aligned} \tag{2}$$

Finally, we derive the probabilistic bound on the overestimation of the Silm-subsketch of the SF_F-sketch.

$$\begin{aligned} Pr[\hat{f_{(e)}} \geqslant f_{(e)} + \epsilon\alpha N] &= Pr[\forall i, A_i[j] \geqslant f_{(e)} + \epsilon\alpha N] \\ &= (Pr[A_i[j] - f_{(e)} \geqslant \epsilon\alpha N])^d \\ &= (Pr[X_{i,(e)}[j] \geqslant \epsilon\alpha N])^d \end{aligned} \tag{3}$$

Substituting the value of $\epsilon\alpha N$ from Equation (2) into the right side of the equation above, we get

$$Pr[\hat{f_{(e)}} \geqslant f_{(e)} + \epsilon\alpha N] \leqslant (Pr[X_{i,(e)}[j] \geqslant \exp E(X_{i,(e)}[j]))^d \tag{4}$$

Applying Markov's Inequality, we get

$$\begin{aligned} Pr[\hat{f_{(e)}} \geqslant f_{(e)} + \epsilon\alpha N] &\leqslant \exp^{-d} \\ &\leqslant \delta \end{aligned} \tag{5}$$

4.2 Derivation of Correct Rate

The *Correct Rate* of a sketch is defined as the expected percentage of items in the given multi-set for which the query response of the sketch contains no error.

In deriving the correct rate of the SF_F-sketch, we make two assumptions: 1) all hash functions are independent; 2) the Fat-subsketch is large enough to have negligible error. Before deriving the correct rate, we first prove the following theorem.

Theorem 1. *In the Slim-subsketch, the value of any given counter is equal to the frequency of the most frequent item that maps to it.*

Proof: We prove this theorem using mathematical induction on the number of insertions, represented by k.

Base Case, k = 0: The theorem clearly holds for the base case because with no insertions, the frequency of the most frequent item is currently 0, which is also the value of all counters.

Induction Hypothesis, k = n: Suppose the statement of the theorem holds true after n insertions.

Induction Step, k = $n+1$: Let $n+1^{\text{st}}$ insertion be of any item e that has previously been inserted a times. Let $\alpha_i(k)$ represent the values of the counter $A_i[h_i(e)]$ after k insertions, where $0 \leqslant i \leqslant d-1$. There are two cases to consider: 1) e was the most frequent item when $k = n$; 2) e was not the most frequent item when $k = n$.

Case 1: If e was the most frequent item when $k = n$, then according to our induction hypotheses, $\alpha_i(n) = a$. After inserting e, it will still be the most frequent item and its frequency increases to $a+1$. The counter $A_i[h_i(e)]$ will be incremented once. Consequently, we get $\alpha_i(n+1) = a+1$. Thus for this case, the theorem statement holds because the value of the counter $A_i[h_i(e)]$ after insertion is still equal to the frequency of the most frequent item, which is e.

Case 2: If e was not the most frequent item when $k = n$, then according to our induction hypotheses, $\alpha_i(n) > a$. After inserting e, it may or may not become the most frequent item. If it becomes the most frequent item, it means that $\alpha_i(n) = a+1$ and as per our SF_F scheme, the counter $A_i[h_i(e)]$ will stay unchanged. Consequently, we get $\alpha_i(n+1) = \alpha_i(n) = a+1$. Thus for this case, the theorem statement again holds because the value of the counter $A_i[h_i(e)]$ after insertion is equal to the frequency of the new most frequent item, which is e.

After inserting e, if it does not become the most frequent item, then it means $\alpha_i(n) > a+1$ and as per our SF_F-sketch scheme, the counter $A_i[h_i(e)]$ will stay unchanged. Consequently, $\alpha_i(n+1) = \alpha_i(n) > a+1$. Thus, the theorem again holds because the value of the counter $A_i[h_i(e)]$ after insertion is still equal to the frequency of the item that was the most frequent after n insertions. □

Next, we derive the correct rate of the SF_F-sketch. Let v be the number of distinct items inserted into the Slim-subsketch and are represented by $e_1, e_2, \ldots, e_v$. Without loss of generality, let the item e_{l+1} be more frequent than e_l, where $1 \leqslant l \leqslant v-1$. Let X be the random variable representing the number of items hashing into the counter $A_i[h_i(e_l)]$ given the item e_l, where $0 \leqslant i \leqslant d-1$ and $1 \leqslant l \leqslant v$. Clearly, $X \sim \text{Binomial}(v-1, 1/w)$.

From Theorem 1, we conclude that if e_l has the highest frequency among all items that map to the given counter $A_i[h_i(e_l)]$, then the query result for e_l will contain no error. Let $\mathbb{E}$ be the event that e_l has the maximum frequency among x items that map to $A_i[h_i(e_l)]$. The probability $P\{\mathbb{E}\}$ is given by the following equation:

$$P\{\mathbb{E}\} = \binom{l-1}{x-1} / \binom{v-1}{x-1} \qquad (\text{where } x \leqslant l)$$

Let P' represent the probability that the query result for e_l from any given counter contains no error. It is given by:

$$P' = \sum_{x=1}^{l} P\{\mathbb{E}\} \times P\{X = x\}$$
$$= \sum_{x=1}^{l} \frac{\binom{l-1}{x-1}}{\binom{v-1}{x-1}} \binom{v-1}{x-1} \left(\frac{1}{w}\right)^{x-1} \left(1 - \frac{1}{w}\right)^{v-x} = \left(1 - \frac{1}{w}\right)^{v-l}$$

As there are d counters, the overall probability that the query result of e_l is correct is given by the following equation.

$$P_{\text{CR}}\{e_l\} = 1 - \left(1 - \left(1 - \frac{1}{w}\right)^{v-l}\right)^{d}$$

The equality above holds when all v items have different frequencies. If two or more items have equal frequencies, the correct rate increases slightly. Consequently, the expected correct rate Cr of the Slim-subsketch is bound by:

$$Cr \geqslant \frac{\sum_{l=1}^{v} P_{\text{CR}}\{e_l\}}{v} = \frac{\sum_{l=1}^{v} \left(1 - \left(1 - (1 - \frac{1}{w})^{v-l}\right)^{d}\right)}{v} \tag{6}$$

Note that in most cases, "=" holds.

4.3 Correct Rate Verification

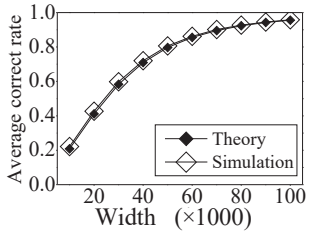


Fig. 5: The Correct Rate

We carried out extensive experiments to compare the theoretical bound on the correct rate derived using Equation 6 with practical one calculated from our simulations. Here, we set $d = 4, v = 100,000, z = 1024$ with width w ranging from 10,000 to 100,000. v represents the number of distinct items. To perform simulation, we update the SF$_{\text{F}}$-sketch with skewed dataset composed of 25M items (*i.e.* 100K distinct items, $1K = 10^3, 1M = 10^3K$). This skewed dataset obeys Zipf distribution with skewness of 0.99. The skewness for this dataset is set to 0.99, a `default value` used in YCSB [33]. As it can be computed very quickly and for multi-byte keys, we randomly choose a family of pairwise independent d Bob Jenkins hash functions for the SF-sketch in this simulation and all the next experiments. Note that we set z to an extremely huge number to make the Fat-subsketch large enough. Figure 5 plots the values of the corresponding correct rate of the two different methods for a varying number of counters per array of the Slim-subsketch. We observe from this figure that the two curves are very close to each other. The values of the calculated practical correct rate are slightly larger than those derived theoretically as the theoretical ones give the lower bound on the correct rate.

4.4 Proof of No Underestimation

Here, we use mathematical induction to prove that the SF$_{\text{F}}$-sketch does not incur any underestimation. Before delving into the proof, we first define some notations. Let $A_i^n[j]$ represent the value of the counter in the j^{th} bucket of the i^{th} array in the Slim-subsketch after n updates, where $1 \leqslant i \leqslant d$, $1 \leqslant j \leqslant w$, and $n \in \mathbb{N}^0$. Similarly, let $B_i^n[(j-1) \times z + k]$ represent the value of the k^{th} counter in the j^{th} bucket of the i^{th} array in the Fat-subsketch after n updates, where $1 \leqslant i \leqslant d$, $1 \leqslant j \leqslant w$, $1 \leqslant k \leqslant z$, and $n \in \mathbb{N}^0$. Furthermore, let $SQ^n(e)$ represent estimate of the frequency of an arbitrary item e calculated from the Slim-subsketch after n updates. Similarly, let $FQ^n(e)$ represent estimate of the frequency of item e calculated from the Fat-subsketch after n updates. As the Fat-subsketch of the SF$_{\text{F}}$-sketch behaves exactly like a CM-sketch for any given item, we directly use the conclusion from [17] that the CM-sketch does not suffer from underestimation, *i.e.*, for any arbitrary item e and $\forall n \in \mathbb{N}^0, FQ^n(e) \geqslant f_{(e)}^n$, where $f_{(e)}^n$ represents the actual frequency of item e after n updates. As querying does not change any counter in either the Fat-subsketch or the Slim-subsketch, we only consider insertion and deletion operations (collectively called updates) in this proof.

Theorem 2. *For updates consisting of insertions and deletions, the SF$_{\text{F}}$-sketch does not incur underestimation. Formally, for any arbitrary item e and $\forall n \in \mathbb{N}^0$*

$$SQ^n(e) \geqslant f_{(e)}^n \tag{7}$$

Proof: We prove this theorem by induction on n, *i.e.*, the number of updates (insertions and deletions).

Base case: The base case occurs when $n = 0$, *i.e.*, no update has yet been done and all counters in both sub-sketches are currently 0. Thus, for any item, the frequency returned by the Slim-subsketch at this point will be 0, *i.e.* for any arbitrary item e, $SQ^0(e) = 0$. As no items have yet been inserted, $f_{(e)}^0 = 0$. Therefore, $SQ^0(e) = f_{(e)}^0$. Thus, the base case of $k = 0$ satisfies Equation (7).

Induction hypothesis: Let for $n = x$, Equation (7) holds true, *i.e.*, for any arbitrary item e, $SQ^x(e) \geqslant f_{(e)}^x$. Our inductive hypothesis is that if Equation (7) holds true for any arbitrary item e when $n \leqslant x$, then it also holds true for that item when $n = x + 1$, *i.e.*, for item e, $SQ^{x+1}(e) \geqslant f_{(e)}^{x+1}$.

Induction step: We use $n = x + 1$ in the inductive step. The value of $SQ^x(e)$ is given as below.

$$SQ^x(e) = A_{i'}^x[h_{i'}(e)] \tag{8}$$

where $i' \in [1, d]$ such that $A_{i'}^x[h_{i'}(e)] = \min_{i=1}^{d} A_i^x[h_i(e)]$. Suppose the item that we are going to insert/delete in the $x + 1^{\text{st}}$ update is e', where e' can also be any arbitrary item. We choose to use e' in the induction step instead of e to make the inductive step generic. Note that the inductive step holds when $e' = e$ as well as when $e' \neq e$. Next, we apply the inductive step separately to the two cases: 1) item e' is inserted; 2) item e' is deleted.

Insertion: When inserting the item e', there are two cases. The first case occurs when $h_{i'}(e) = h_{i'}(e')$ regardless of whether $e = e'$ or $e \neq e'$. The second case occurs when $h_{i'}(e) \neq h_{i'}(e')$. Note that this second case occurs only when $e \neq e'$.

For the first case, where $h_{i'}(e) = h_{i'}(e')$, the counter $A_{i'}[h_{i'}(e)]$ may be incremented if $A_{i'}^x[h_{i'}(e)] < \min_{i=1}^{d} B_i^{x+1}[g_i(e')]$ ($g_i(.)$ is defined as in Equation 1). In this case,

$$A_{i'}^{x+1}[h_{i'}(e)] = A_{i'}^x[h_{i'}(e)] + 1$$
$$= SQ^x(e) + 1$$
$$\geqslant f_{(e)}^x + 1$$

If $e \neq e'$, then $f_{(e)}^{x} + 1 > f_{(e)}^{x+1}$, otherwise $f_{(e)}^{x} + 1 = f_{(e)}^{x+1}$. Combining these two, we get $f_{(e)}^{x} + 1 \geqslant f_{(e)}^{x+1}$. Substituting this inequality in the equation above, we get

$$A_{i'}^{x+1}[h_{i'}(e)] \geqslant f_{(e)}^{x} + 1 \geqslant f_{(e)}^{x+1}$$

As $A_{i'}^{x+1}[h_{i'}(e)] = SQ^{x+1}(e)$, we get

$$SQ^{x+1}(e) \geqslant f_{(e)}^{x+1}$$

Thus, the inductive hypothesis holds true for the first case.

For the second case, where $h_{i'}(e) \neq h_{i'}(e')$, the counter $A_{i'}[h_{i'}(e)]$ stays unchanged, *i.e.*, $A_{i'}^{x+1}[h_{i'}(e)] = A_{i'}^{x}[h_{i'}(e)]$, which means that $SQ^{x+1}(e) = SQ^{x}(e)$. This further implies that $SQ^{x+1}(e) \geqslant f_{(e)}^{x}$. As for this case, $e \neq e'$, $f_{(e)}^{x} = f_{(e)}^{x+1}$, we get $SQ^{x+1}(e) \geqslant f_{(e)}^{x+1}$. Thus, the inductive hypothesis holds true for the second case as well.

Deletion: According to the deletion operation of the SF_F-sketch, a deletion after x updates may cause a counter $A_i[j]$ in the Slim-subsketch to either stay unchanged or be set to $\max_{k=1}^{z} B_i^{x+1}[(h_i(e)-1) \times z + k]$. For each $i \in [1, d]$, if $h_i(e) = h_i(e')$ and $A_i^{x}[h_i(e)] > \max_{k=1}^{z} B_i^{x+1}[(h_i(e)-1) \times z + k]$, we have

$$\begin{aligned} A_i^{x+1}[h_i(e)] &= \max_{k=1}^{z} B_i^{x+1}[(h_i(e)-1) \times z + k] \\ &\geqslant B_i^{x+1}[g_i(e)] \\ &\geqslant \min_{i'=1}^{d} B_{i'}^{x+1}[g_{i'}(e)] \\ &= FQ^{x+1}(e) \\ &\geqslant f_{(e)}^{x+1} \end{aligned} \tag{9}$$

Otherwise, we have:

$$\begin{aligned} A_i^{x+1}[h_i(e)] &= A_i^{x}[h_i(e)] \geqslant \min_{i'=1}^{d} A_{i'}^{x}[h_{i'}(e)] \\ &= SQ^{x}(e) \geqslant f_{(e)}^{x} \geqslant f_{(e)}^{x+1} \end{aligned} \tag{10}$$

Thus, each counter to which e maps stays larger than the actual frequency of e on deletion after x updates, which in turn implies that $SQ^{x+1}(e) \geqslant f_{(e)}^{x+1}$ and the inductive hypothesis holds true. □

5 IMPLEMENTATION

In this section, we describe our implementation of the sketches on two different computing platforms namely CPU and GPU. We extensively tested and evaluated the SF-sketch and compared its performance with prior sketches on these two platforms. Next, we first describe our implementation on the CPU platform and then describe our implementation on the GPU platform.

5.1 CPU Implementation

Our CPU platform comprised a machine with dual 6-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62 GB total system memory. Each CPU has three levels of cache memory: L1, L2, and L3. L1 cache is comprised of two 32KB caches, where one cache acts as the data cache and the other acts as the instruction cache. L2 cache is a single 256KB cache and L3 cache is a single 15MB cache. To evaluate the schemes in different types of settings, our implementations on the CPU platform include both single-thread implementation as well as multi-thread implementation. We used C++ as the programming language. In single-thread implementation, for each sketch, we implemented the entire insertion, deletion, and query process within a single thread. In multi-thread implementation, we run each query in a dedicated thread and process it completely inside that thread, observing near-linear growth in query speed with the increase in the number of threads. We will present the experimental results on query speed with multiple threads in more detail in Section 6.3.3.

5.2 SIMD Implementation

In the SF_F-sketch, the key operation of producing the Slim-subsketch is to select the maximum counter in each bucket. As such selections in different buckets of the Fat-subsketch do not affect each other, we can use SIMD (Single Instruction Multiple Data) [19] operations to concurrently process different buckets, so as to significantly boost the speed. As shown in Figure 6, the i^{th} counters of different buckets can be organized into the same vector register for SIMD operation. All the counters in the i^{th} bucket are loaded in the i^{th} cells of vector registers. For example, $A_i(1 \leqslant i \leqslant 16)$ are loaded in the first cells of vector registers, while $B_i(1 \leqslant i \leqslant 16)$ are loaded in the second cells. Parallel comparisons are performed on multiple counters using the vector registers. The number of parallel comparisons is determined by the vector size and counter size. The vector size can be 64 bits (register MMX), 128 bits (XMM), 256 bits (YMM), or 512 bits (ZMM). We choose XMM registers and SSE4.1 (CPUID Flag) instruction set as an example, to illustrate how the producing speed can be accelerated. Given 128-bit vectors of XMM, if 32-bit counters are chosen, we can fit 4 counters into each vector and perform 4 comparisons in parallel, achieving $4\times$ speedup. For example, in Figure 6, A, B, C, D stands for 4 buckets in the Fat-subsketch stored in memory sequentially, and each contains 16 counters. Using instructions "pmaxud, xmm, xmm", the bigger counters are loaded in M_i. In this figure, only 3 XMM registers are used, knowing that more registers can be harnessed to preload the remaining $16 - 2 = 14$ counters for further speedup.

5.3 GPU Implementation

As GPUs have seen wide acceptance for high-speed data processing, we implemented our sketches on GPUs as well. For these implementations, we employ the basic architecture of GAMT [34]. More specifically, we evaluated the sketches on GPU platform using CUDA 5.0 architecture. We performed our experiments on the same server and an NVIDIA GPU (Tesla C2075, 1147 MHz, 5376 MB device memory, 448 CUDA cores). We implemented our sketches on GPU using two prevalent techniques: batch processing and multi-stream pipelining. Next, we describe our implementations for these two techniques.

5.3.1 Batch Processing

Our system architecture is based on CUDA [35], the well-known parallel computing platform created by NVIDIA. In our implementation, a typical query cycle proceeds in the following three steps: (1) copy the incoming queries from the CPU to the GPU, (2) execute the query kernel, and (3) copy the result from the GPU back to the CPU. A kernel in CUDA is a function that is called on CPU but executed on GPU. A query kernel is configured with a series of thread blocks, where each block is comprised of a group of working threads. As GPU chips have hundreds and even thousands

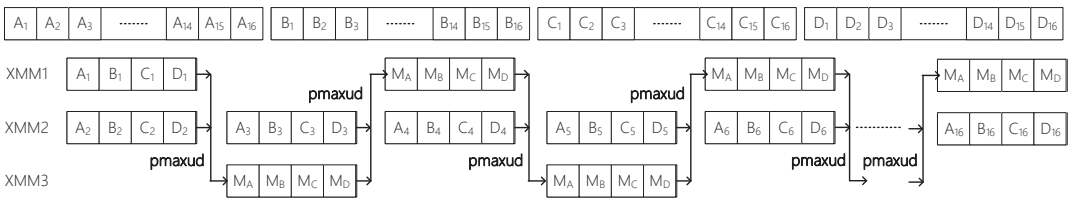


Fig. 6: SIMD speedup implementation.

of cores, batch processing is needed to accelerate GPU-based implementations. Each batch is first filled with a group of independent queries, and then transferred to and executed on the GPU, *i.e.*, as soon as a query arrives, it is buffered until there are enough queries to fill the current batch of queries before transferring the batch to GPU for processing by the query kernels.

5.3.2 Multi-Stream Pipeline

As discussed earlier, batch processing is required to take advantage of the massive parallelization that GPU enables. However, waiting for enough queries to fill a batch before sending the batch to GPU results in unnecessary delays. Furthermore, while a large batch does boost the throughput of the GPU, it increases the waiting time before a batch fills and is transferred to GPU for processing. This means that the query that arrived at the start of the current batch will experience significant latency before it is processed. To resolve this throughput-latency dilemma, we utilize the multi-stream technique featured in NVIDIA Fermi GPU architecture [34], [36].

6 Experimental Results

We conducted extensive experiments to evaluate the performance of our SF_F-sketch in terms of *accuracy* and *speed*. Onwards, we will refer to the SF_F-sketch as simply the SF-sketch. For comparison, we also implemented and evaluated the performance of four well known sketches, namely the C-sketch [16], the CM-sketch [17], the CU-sketch [18], and the A-sketch [7]. As mentioned in Section 2, when the A-sketch is applied to the CM-sketch, the accuracy stays almost unchanged using 32 items in the filter as the authors recommended, which is verified by our subsequent experimental figures. Therefore, we do not discuss more about the A-sketch in this section. In addition, note that *the CU-sketch does NOT support deletions and that the C-sketch suffers from both overestimation and underestimation.*

6.1 Experimental Setup

Datasets: We use three types of datasets: real world traffic, uniform dataset, and skewed dataset. The real world network traffic trace is captured by the main gateway of our campus, while the uniform and the skewed datasets are generated by the well known YCSB [33]. Throughout the experiments, we use skewed datasets that obey Zipf distribution. We keep the skewness of our skewed dataset equal to 0.99, which is the `default value` used in YCSB. We use Memcached [37] to record the real frequency of each item to establish the ground truth.

CPU and GPU Setup: Our CPU platform comprised a machine with dual 6-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62 GB total system memory. Each CPU has three levels of cache memory: L1, L2, and L3. L1 cache is comprised of two 32KB caches, where one cache acts as the data cache and the other acts as the instruction cache. L2 cache is a single 256KB cache and L3 cache is a single 15MB cache. We performed our experiments on a NVIDIA GPU (Tesla C2075, 1147 MHz, 5376 MB device memory, 448 CUDA cores).

Experimental Comparison: As the memory of the monitor is cheap and large enough, we assign the same size of memory for the Slim-subsketch and the state-of-the-art sketches both of which will be transmitted to the collector. The Fat-subsketch uses 16 counters in each bucket, and thus is 16 times larger than the Slim-subsketch and the state-of-the-art sketches. For update experiments, we compare them by varying item frequencies and operation size, *i.e.*, the number of insertion and deletion operations.

6.2 Experiments on Accuracy

We use *absolute error* (AE) and *relative error* (RE) to quantify the accuracy of sketches. Let f_e be the actual frequency of an item e and $\hat{f}_e$ be the estimate of the frequency returned by the sketch, the `Absolute Error` is defined as $|\hat{f}_e - f_e|$, while the `Relative Error` is defined as the ratio $|\hat{f}_e - f_e|/f_e$. To evaluate accuracy, we inserted 10M items (100K distinct items for uniform datasets, 100K distinct items for skewed datasets and approximately 233K for real datasets, $1K = 10^3$) and fixed parameter setting (d = 4, w = 40000, $z = 2^4$ = 16). We calculated absolute error and relative errors of sketches *by querying each distinct item once*. We also conducted experiments to quantify the effect of system parameters on the performance of the sketches.

6.2.1 Uniform Workload

Relative Error CDF: *Our experimental results show that the percentage of items for which the relative error of our SF-sketch is less than 1% is 71.36%, which is 5.16, 2.45 and 2.27 times higher than the corresponding percentages for the C-, CM- and CU-sketches, respectively.* Figure 7 reports the empirical cumulative distribution function (CDF) of relative error for the 100K distinct items after a total of 10M ($= 10 \times 10^6$) insertions. Specifically, we first inserted the 100K distinct items for a total of 10M times such that the probability of occurrence for each item was uniformly distributed, and then calculated the relative errors in the estimates of the frequencies of those 100K distinct items. In this way, we got 100K values of relative error for each of the five sketches (the C-, CM-, A-, CU- and SF-sketches). We plotted a CDF using the 100K relative error values for each sketch. Figure 7 shows that the CDF of the SF-sketch is not only higher than that of the other four sketches but also ascends sharply near the relative error of 0. This indicates that the relative error

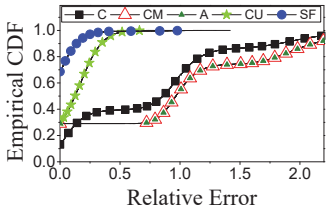


Fig. 7: CDF of Uniform RE.

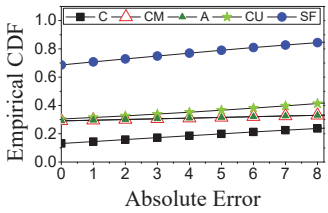


Fig. 8: CDF of Uniform AE.

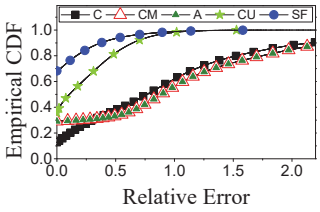


Fig. 9: CDF of Zipf RE.

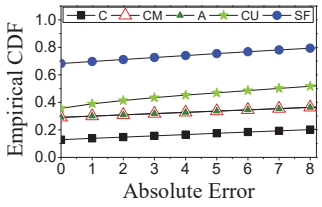


Fig. 10: CDF of Zipf AE.

in the estimate of the frequencies of most items, calculated from the SF-sketch, is very close to 0.

Absolute Error CDF: *Our experimental results show that the percentage of items for which the absolute error of our SF-sketch is equal to 0 is 70.47%, which is 5.37, 2.32 and 2.42 times higher than the corresponding percentages for the C-, CM- and CU-sketches, respectively.* Figure 8 reports the empirical cumulative distribution function (CDF) of absolute error for the 100K distinct items after a total of 10M ($= 10 \times 10^6$) insertions.

6.2.2 Skewed Workload

For skewed workloads, we performed exactly the same experiments as for the uniform workloads. The experimental results are shown in Figures 9 and 10. The trends in these figures for the skewed distribution are similar to what we observed for the uniform distribution. Therefore, for the sake of brevity, next, we concisely report the results without describing again how the experiments were conducted.

Relative Error CDF: *Our experimental results, reported in Figure 9, show that in the case of skewed workload, the percentage of items for which the relative error of our SF-sketch is less than 1% is 70.23%, which is 5.4, 2.4 and 1.9 times higher than the corresponding percentages for the C-, CM- and CU-sketches, respectively.*

Absolute Error CDF: *Our experimental results, reported in Figure 9, show that in the case of skewed workload, the percentage of items for which the absolute error of our SF-sketch is equal to 0 is 70.23%, which is 5.5, 2.4 and 2.0 times higher than the corresponding percentages for the C-, CM- and CU-sketches, respectively.*

6.2.3 Real Traffic

We also used real traffic to evaluate the accuracy of sketches. We have 10M real traffic instances and regard the traffic with the same destination IP address to belong to the same flow. Using this definition of a flow, there are about 233K flows in our datasets, and the size distribution of flows is biased with an expected value of 42.87 and a variance of 1457342.

Relative Error CDF: *Our experimental results, reported in Figure 11, show that in the case of real world traffic, after a total of 10M insertions of the 233K distinct items, 36.51% items with our SF-sketch have relative error less than 1%, while the percentages of items with the C-, CM- and CU-sketches are 6.87%, 2.56% and 11.73%, respectively.* The relative error seems larger than anticipated because about 43.86% of flows have only 1 packet in the real traffic, while 147 flows have no less than 10,000 packets. If a flow with only 1 packet is estimated to 101, the relative error will be 10000% ($(101-1)/1$).

Absolute Error CDF: *Our experimental results, reported in Figure 12, show that in the case of real world traffic, after a total of 10M insertions of the 233K distinct items, 36.47% items with our SF-sketch have no error, while the percentages of items with the C-, CM- and CU-sketches are 5.88%, 1.69% and 11.54%, respectively.*

6.2.4 Sketch Parameters

Next, we evaluate the effect of changing the system parameters d (the number of arrays) and w (the number of buckets per array) on the accuracy of the sketches. In each experiment to evaluate the effect of system parameters, we insert the 100K distinct items 10M times.

Accuracy vs. *w*: *Our experimental results show that the CU-sketch requires 1.5 times more memory compared to the SF-sketch to achieve close to 1% average relative error.* Figure 13 plots the average relative error by varying the number of buckets per array with d fixed at 5. We observe from this figure that increasing the number of buckets per array reduces the average relative error for each sketch. However, we observe that at 30K buckets per array, the average relative error of our SF-sketch reduces to a very small value of just 0.047. On the contrary, the CU-sketch requires 50K buckets per array to achieve an average relative error of 0.049, but note that it does not support deletions. The C-, CM- and A-sketches did not achieve close to 0 average relative errors in our experiments.

Accuracy vs. *d*: *Our experimental results show that our SF-sketch achieves an average relative error of 5.6% using only 3 arrays whereas the CU-sketch takes 6 arrays to come close to the error of the SF-sketch and achieves the average relative error of 7.1%.* This shows that compared to the CU-sketch, at this error rate, the SF-sketch takes only half as much memory. Figure 14 plots the average relative error by varying d with w fixed at $40K$. We observe from this figure that using 6 arrays, the SF-sketch achieves an average relative error of 1.9%. We also observe that increasing the number of arrays reduces the average relative error for all the sketches.

6.3 Experiments on Speed

Next, we evaluate the production speed, update speed and query speed of the sketches. For GPU platform, the query speed is the throughput, and we also need to evaluate the latency, which is measured in microseconds and quantifies the time duration between submitting a query and receiving the response. We observed from our experiments that the query speeds of the CM-, CU- and SF-sketches have little differences. This observation was expected because the query operations of these sketches are almost the same. For this reason, we only present the experimental results of the query speed of the SF-sketch on both multi-core CPU and GPU platforms.

6.3.1 Production Speed with/without SIMD

Our experimental results show that the production speed of the SF-sketch using SIMD is $6.5 \sim 8.5$ *times faster than that*

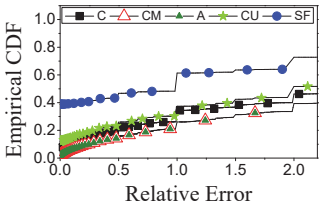


Fig. 11: CDF of Real RE.

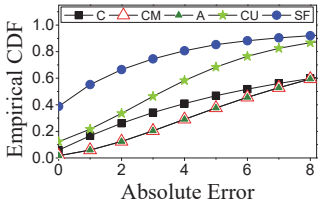


Fig. 12: CDF of Real AE.

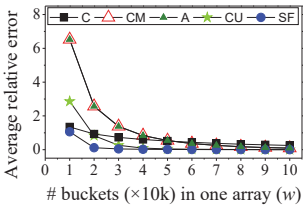


Fig. 13: Accuracy vs. w.

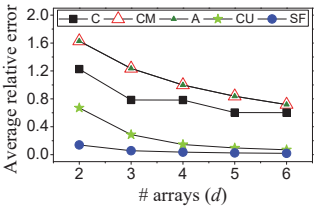


Fig. 14: Accuracy vs. d.

without SIMD. As shown in Figure 15, the x-axis is $d \times w$, where d is the number of arrays and w is the number of buckets in each array. Each bucket includes 16 counters, and each counter is 32-bit long to achieve word alignment, while each counter in the corresponding Slim-subsketch and prior sketches has 24 bits. The y-axis is the production speed, which indicates how many Gigabytes (GB) in the Fat-subsketch are processed in one second. As the memory size of the Fat-subsketch increases, the production speed drops gradually. Specifically, when $d \times w$ is $40K \times 4$, the production time is only 2 milliseconds when using SIMD and only one CPU core, which is small enough to be ignored in most applications.

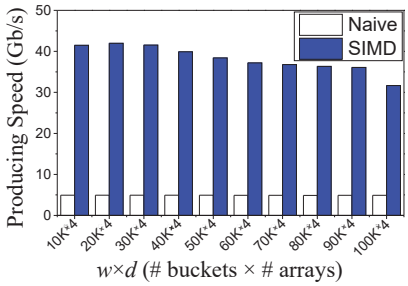


Fig. 15: The production speed of the Slim-subsketch from the Fat-subsketch.

6.3.2 Update Speed and Query Speed

Our experimental results show that the insertion and deletion speeds of the SF-sketch are 12.7% *and* 11.9% *slower than those of the CM-sketch, respectively, the query speeds of the SF-, CM- and CU-sketches are almost the same, and* 1.74 *times faster than the C-sketch.* Our experimental results show that the CM-sketch achieves the fastest insertion and deletion speeds. Thus, we mainly compare our SF-sketch with the CM-sketch in terms of update/query speed (measured in Mops, million operations per second). Figure 16 plots the update and query speeds of the SF-, CM-, CU- and C-sketches. We conducted experiments using many datasets and observed similar results. Figure 16 shows plots for 10 randomly chosen datasets, based on each of which we generate 10M inserts, 10M deletes (except the C-sketch), and 10M queries for each sketch. We use the same sketch parameters as used in Figure 11. Our results also show that the update speed of the SF-sketch is a little slower than the CM-sketch, because the sketch used for update is the Fat-subsketch, which is 16 times larger than the Slim-subsketch and the CM-sketch. The query speeds of the SF-, CM- and CU-sketches are the same because their query operations are the same; the query speed of the C-sketch is slower as it needs twice the number of hash computations and getting the median of d counters is time consuming than getting the minimum of them. In this experiment, we only use one CPU core to perform updates and queries. One can achieve much faster update speed using multiple cores or GPU.

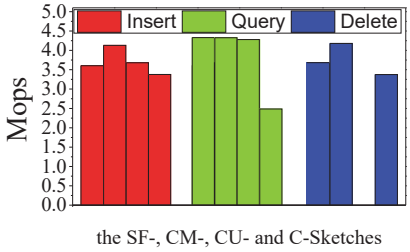


Fig. 16: The update and query speed of sketches.

6.3.3 Multi-core CPU Platform

As CPUs are multi-core in today's control centers. Thus, the query algorithm for the SF-sketch can be run in parallel. Next, we examine the query throughput with increasing number of threads, which will help us understand how the number of cores can affect the performance of query throughput. Since a single-core experiment for comparison between sketches can be found in Section 6.3.2, we've omitted the comparison between the SF-, CM-, CU- and C-sketches here as well as in the next Section 6.3.4.

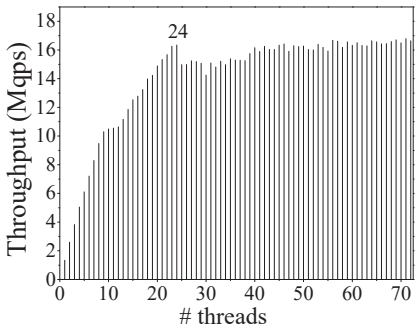


Fig. 17: Query throughput vs. # of threads.

Our experimental results show that the SF-sketch experienced a throughput gain of about 650K queries per second per thread up to 24 threads. Figure 17 plots the throughput vs. the number of threads for the SF-sketch. We observe from this figure that the SF-sketch achieves a throughput of about 1.34M queries per second with a single thread. For this experiment, we performed 10M queries. Using 24 threads, it achieves a throughput of about 16.3M queries per second. We further observed that increasing the number of threads beyond 24 brought little improvement because our CPU has 6×2 cores,

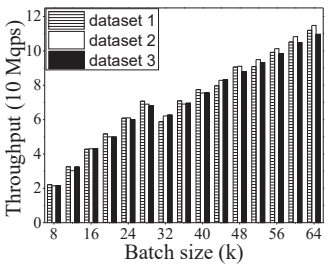


Fig. 18: Query throughput vs. batch size.

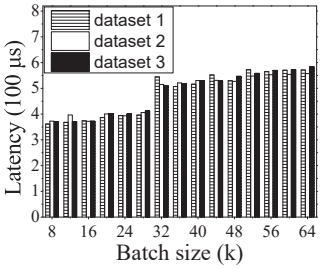


Fig. 19: GPU query latency vs. batch size.

which support $6 \times 2 \times 2 = 24$ threads. This suggests that the query speed of our SF-sketch increases linearly with the number of CPU cores.

6.3.4 GPU Platform

Our experimental results for three different datasets show that the query speed in GPU increases with the increase in the batch size. As shown in Figure 18, for the batch size of 20K queries, the query speed is around 50 million queries per second (Mqps). With increase in the batch size, such as 64K queries per batch, the SF-sketch reaches a query speed greater than 110 Mqps.

Our experimental results for three different datasets show that for the SF-sketch, to reduce latency, the batch size of 28K is the most optimal for our experimental setup. Figure 19 shows that the average query latency of the SF-sketch is below $410\ \mu s$ for batch sizes $\leqslant 28K$. For batch sizes $\geqslant 32k$, the latency increases to $511 \sim 584\ \mu s$.

7 Conclusion

In this paper, we proposed a new sketch for data streams, namely the SF-sketch, which achieves up to 33.1 times higher accuracy compared to the CM-sketch while keeping the update and query speeds comparable with the CM-sketch. The key idea behind our proposed SF-sketch is to use two separate sketches, one is called the Fat-subsketch and the other is called the Slim-subsketch. The Fat-subsketch is used to perform updates and to periodically produce the Slim-subsketch within a few milliseconds, which is periodically transferred to the remote collector for answering queries quickly and accurately. To evaluate and compare the performance of our proposed SF-sketch, we conducted extensive experiments on multi-core CPU and GPU platforms. Our experimental results show that our SF-sketch significantly outperforms the state-of-the-art in terms of accuracy.

Acknowledgments

This work is partially supported by Primary Research & Development Plan of China (2016YFB1000304, 2018YFB1004403, and 2018YFE0207600), NSFC (Grants No. 61571352, 61672061, U1536202, and U1736216), National Science Foundation of USA (Grants No. CNS-1616317 and CNS-1616273), Shaanxi Science & Technology Coordination & Innovation Project (2016KTZDGY05-07-01), the Fundamental Research Funds for the Central Universities and the Innovation Fund of Xidian University. Yulong Shen (ylshen@mail.xidian.edu.cn) and Tong Yang (yangtongemail@gmail.com) are the corresponding authors.

References

[1] "Source code of SF sketches," https://github.com/paper2017/SF-sketch.

[2] T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie, "Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 2017, pp. 103–106.

[3] C. C. Aggarwal and S. Y. Philip, "On classification of high-cardinality data streams." in *SDM*, vol. 10. SIAM, 2010, pp. 802–813.

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

[5] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1530–1541, 2008.

[6] Z. Liu, A. Manousis, and et al., "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proc. ACM SIGCOMM*, 2016.

[7] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. ACM SIGMOD*, 2016.

[8] D. Thomas, R. Bordawekar, and et al., "On efficient query processing of stream counts on the cell processor," in *Proc. IEEE ICDE*, 2009.

[9] 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*. VLDB Endowment, 2005, pp. 13–24.

[10] A. Margara, J. Urbani, F. van Harmelen, and H. E. Bal, "Streaming the web," *Journal of Web Semantics*, vol. 25, pp. 24–44, 2014.

[11] H. Zhan, G. Gomes, X. S. Li, K. Madduri, A. Sim, and K. Wu, "Consensus ensemble system for traffic flow prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 12, pp. 3903–3914, 2018.

[12] C. Baquero, P. S. Almeida, R. Menezes, and P. Jesus, "Extrema propagation: Fast distributed estimation of sums and network sizes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 668–675, 2012.

[13] D. Tong and V. K. Prasanna, "Sketch acceleration on fpga and its applications in network anomaly detection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 929–942, 2018.

[14] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 561–575.

[15] Q. Huang, P. P. Lee, and Y. Bao, "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 2018, pp. 576–590.

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

[17] 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.

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

[19] "Introduction to intel advanced vector extensions," https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions.

[20] F. Rusu and A. Dobra, "Statistical analysis of sketch estimators," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM, 2007, pp. 187–198.

[21] ——, "Sketches for size of join estimation," *ACM Transactions on Database Systems (TODS)*, vol. 33, no. 3, p. 15, 2008.

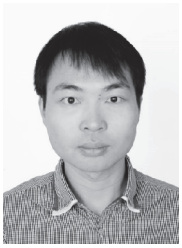
[22] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, "Diamond sketch: Accurate per-flow measurement for big streaming data," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[23] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[24] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD*, 2003, pp. 241–252.

[25] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J.-L. Larriba-Pey, "Dynamic count filters," *ACM SIGMOD Record*, pp. 26–32, 2006.

[26] P. Pandey, M. A. Bender, R. Johnson, and R. Patro, "A general-purpose counting filter: Making every bit count," in *Proc. ACM SIGMOD 2017*, 2017, pp. 775–787.
[27] T. Yang, A. X. Liu, and et al., "A shifting bloom filter framework for set queries," in *Proc. VLDB*, 2016.
[28] M. A. Bender, M. Farach-Colton, and et al., "Don't thrash: how to cache your hash on flash," *Proceedings of the VLDB Endowment*, vol. 5, no. 11, pp. 1627–1637, 2012.
[29] M. Goswami, D. Medjedovic, E. Mekic, and P. Pandey, "Buffered Count-Min Sketch on SSD: Theory and Experiments," in *ESA '18*, 2018, p. 15.
[30] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: a better netflow for data centers," in *Proc. USENIX NSDI*, 2016, pp. 311–324.
[31] Y. Zhu, N. Kang, and et al., "Packet-level telemetry in large datacenter networks," in *Proc. ACM SIGCOMM*, 2015.
[32] N. Handigol, B. Heller, and et al., "I know what your packet did last hop: Using packet histories to troubleshoot networks." in *NSDI*, vol. 14, 2014, pp. 71–85.
[33] B. F. Cooper, A. Silberstein, and et al., "Benchmarking cloud serving systems with YCSB," in *Proc. ACM SOCC*, 2010.
[34] Y. Li, D. Zhang, A. X. Liu, and J. Zheng, "GAMT: a fast and scalable ip lookup engine for gpu-based software routers," in *Proc. IEEE/ACM ANCS*, 2013.
[35] *NVIDIA CUDA C Best Practices Guide, Version 5.0*, NVIDIA Corporation, Oct. 2012.
[36] Y. Wang, Y. Zu, and et al., "Wire speed name lookup: A gpu-based approach," in *Proc. USENIX NSDI*, 2013, pp. 199–212.
[37] "Memcached - a distributed memory object caching system," http://memcached.org.



Lingtong Liu is currently working toward the PhD degree in the School of Computer Science and Technology, Xidan University. His PhD work mainly concerns on data sketches, IP lookups, and cloud security.

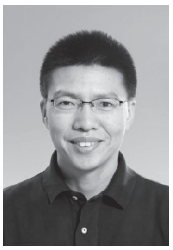
Yulong Shen received the B.S. and M.S. degrees in computer science and PhD degree in cryptography from Xidian University, Xi'an, China, in 2002, 2005, and 2008, respectively. He is currently a Professor with the School of Computer Science and Technology, Xidian University, where he is also an Associate Director of the Shaanxi Key Laboratory of Network and System Security and a member of the State Key Laboratory of Integrated Services Networks. His research interests include wireless network security and cloud computing security. He has also served on the technical program committees of several international conferences, including ICEBE, INCoS, CIS, and SOWN.

Yibo Yan is a postgraduate at Peking University, advised by Tong Yang. His research interests include indexing, data sketches, and data stream processing systems.

Tong Yang received his PhD degree in Computer Science from Tsinghua University in 2013. He visited Institute of Computing Technology, Chinese Academy of Sciences (CAS). Now he is a research assistant professor in Computer Science Department, Peking University. His research interests include network measurements, sketches, IP lookups, Bloom filters, sketches and KV stores. He published papers in SIGCOMM, SIGKDD, SIGMOD, SIGCOMM CCR, VLDB, ATC, ToN, ICDE, INFOCOM, *etc.*

Muhammad Shahzad received the PhD degree in computer science from Michigan State University in 2015. He is currently an Assistant Professor with the Department of Computer Science, North Carolina State University, USA. His research interests include design, analysis, measurement, and modeling of networking and security systems. He received the 2015 Outstanding Graduate Student Award, the 2015 Fitch Beach Award, and the 2012 Outstanding Student Leader Award at Michigan State University.

Bin Cui is a professor in the School of EECS and director of the Institute of Network Computing and Information Systems, Peking University. His research interests include database systems and data mining. He has published more than 100 research papers, and is the winner of the Microsoft Young Professorship award (MSRA 2008), and CCF Young Scientist award (2009).



Gaogang Xie received the PhD degree in computer science from Hunan University, Changsha, China, in 2002. He is currently a Professor and the Director of the Network Technology Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His research interests include programmable virtual routers, future Internet architecture, and Internet measurement.