ABC: a Practicable Sketch Framework for Non-uniform Multisets

Junzhi Gong*, Tong Yang*†, Yang Zhou*, Dongsheng Yang*, Shigang Chen‡, Bin Cui*, Xiaoming Li*
*Department of Computer Science, Peking University, China
†Collaborative Innovation Center of High Performance Computing, NUDT, China
‡Department of Computer & Information of Science & Engineering, University of Florida, USA

***Abstract*—Sketch is a data structure used to record frequencies of items in a multiset, which is widely used in data streams, data graph, distributed datasets processing, *etc*. It works with small memory usage and a high speed at the cost of a slight inaccuracy. In practice, frequencies of items in many datasets are non-uniformly distributed. Unfortunately, existing sketches can hardly work well on non-uniform datasets. To address this issue, we propose a new sketch framework, namely ABC framework, which can be applied to most existing sketches and can significantly improve the accuracy on non-uniform datasets. The key idea behind our framework is that when a counter overflows, it makes use of the space from the adjacent counters by operations of `bits-borrowing` and `combination`. Extensive experimental results show that our ABC framework improves the accuracy by 4.10 times and 4.49 times in average, respectively. A demo and all the related source codes are available on our homepage [1].**

***Index Terms*—Data Structure, Sketch, Data Streams, Non-uniform Datasets**

I. Introduction

A sketch is a probabilistic data structure that stores the frequencies of items in a multiset, and provides an estimated frequency of any item in or not in the multiset. Among all the existing methods for approximate query processing, sketches have a limited size and a faster processing speed than most of other methods, on account of its probabilistic feature and the updating method which are largely independent of the current state of the summary. Furthermore, the sketch also has a high accuracy for queries. A sketch is often associated with three operations: insertion, deletion and query. An insertion operation of an item will increase the frequency of the item stored in the sketch by 1, and correspondingly a deletion operation of an item will decrease the frequency of the item by 1, and a query operation of an item will report the current approximate frequency of the item.

Sketches have been originally used to approximately query the streaming data [2], [3], [4], [5], [6], [7], [8], [9]. Sketches can also be used to query the frequency of finite multisets, such as TreeSketches [10] and Tuple Graph synopses [11] for data graph, the well known algorithm BIRCH [12] for finding useful patterns in large datasets, the approximation of join and related problems using AGMS [13] and Fast-Count [14], mining top-K frequent items in databases [15], outlier detection in high-dimensional data based on AMS sketch [16], information aggregation for sensor database [17], [18], information theoretic feature selection [19], graph streams [20], and more [21], [22], [23]. In this paper, we focus on sketches used for querying the frequency of items in a multiset.

With regard to the performance of the sketch, we focus on four key metrics: accuracy, update speed, query speed, and the size of memory usage the sketch requires. Among these four metrics, we mainly focus on the improvement of accuracy, which indicates the relative error of the frequency of each item on average. Currently, different sketches have different advantages and disadvantages, and are suitable for different application scenarios. Existing sketches can work well for the multisets following uniform distribution. However, frequencies of items in most real datasets are non-uniformly distributed [24], [7], [25]. In most real datasets, most items have small frequencies while only a few items have large frequencies. Unfortunately, existing sketches can hardly work well on non-uniform datasets because of two reasons: 1) If the number of bits is determined by the maximum frequency, most counters always have small values, and this is a waste of memory; 2) If fewer bits are used for each counter, counter overflow will happen, and new errors such as under-estimation errors will occur. What is worse, *hot items* (items with large frequencies) can hardly get accurate estimations in this case. An elegant scheme should achieve accurate estimation for both hot items and *cold items* (items with small frequencies). In sum, *only when high memory efficiency is achieved, can the accuracy be optimized*.

To improve the memory efficiency of sketches for non-uniform dataset, one straightforward way is to use additional counters and additional hashings. However, such a method needs more hash computations and memory accesses, and thus will badly degrade the processing speed (*i.e.*, update speed) for high-speed data streams. Actually, the insertion speed is at least as important as the accuracy in practice, because if the inserting process is not fast enough, data loss will happen, which is hardly acceptable in practice. What is worse, hot items will collide in the additional counters, and the accuracy of hot items will be badly degraded, while hot items are often

*Corresponding author: Tong Yang (Email: yang.tong@pku.edu.cn). This work was done by Junzhi Gong, Yang Zhou, and Dongsheng Yang under the guidance of their mentor: Tong Yang. This work is partially supported by Primary Research & Development Plan of China (2016YFB1000304), National Basic Research Program of China (2014CB340400), NSFC (61472009, 61672061), the Open Project Funding of CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences.

more important than cold items, especially in detections of top-K hot items and heavy hitters.

The design goal of this paper is to improve the memory efficiency while keeping the processing speed of sketches unchanged. Towards this goal, we propose a universal sketch framework, namely Adjacent Borrow and Carry (ABC), achieving higher accuracy while keeping processing speed unchanged. Our ABC framework is based on *our observation*: extensive experiments on real datasets show that when a hot item is mapped to a counter C, the adjacent counters of C are often empty or mapped by cold items. In other words, the adjacent counters have available memory (*i.e.*, bits). The reason behind is that cold items are often much more than hot items in practice. Based on this observation, we propose the *bits-borrowing* technique, *i.e.*, to borrow bits from the adjacent counters when a mapped counter overflows. In the worst case, the adjacent counters have no available bits, we propose the *combination* technique, *i.e.*, to *combine* the mapped counter and its adjacent counter into a big counter, to significantly enlarge the expression range than that of the original small counters (*i.e.*, from 255 to 65535 when each counter is 8 bits). Even in the worst case, two hot items are mapped into two adjacent counters, but this situation can hardly happen in all the d hash mappings, while existing sketches perform d hashing, and report the counter with slightest collisions. By borrowing bits or combining small counters, the memory efficiency of sketches is significantly improved, and thus the accuracy is optimized. Meanwhile, during this process, no additional hash computation or memory accesses is needed, and thus the processing speed is not degraded using ABC. Furthermore, our ABC sketch framework can be applied to most existing sketches (*e.g.*, the Count sketch [26], the Count-min sketch [27], the CU sketch [24], the CML sketch [28]), indicating the good generality of our framework.

In this paper, we have made the following key contributions:

- First, we propose a novel sketch framework named the ABC framework, which significantly improves the accuracy of sketches, keeping the processing speed unchanged.
- Second, we apply our ABC framework to two typical sketches: the CM sketch and the CU sketch as case studies. Our framework can also be applied to other sketches, including the Count sketch, the CML sketch, *etc.*
- Third, we have conducted extensive experiments to verify our formulas and compare the performance of sketches before and after using our ABC frameworks.

II. Related Work

The pioneering work of sketches is the Count sketch (C sketch for short) proposed by Charikar *et al.* [26]. A C sketch consists of d arrays, and each array A_i is associated with two hash functions $h_i(.)$ and $g_i(.)$ ($1 \leqslant h_i(.)\%w \leqslant w$, and *we will omit '$\%w$' in the rest of this paper for conciseness*). $g_i(.)$ maps each item to -1 or $+1$ with the same probability. When inserting an item e, it calculates all hash functions and adds $g_i(e)$ to the counters $\mathrm{A}_i[h_i(e)]$ for each i ($1 \leqslant i \leqslant d$). When querying an item e, it just reports the median of $\mathrm{A}_1[h_1(e)]\times g_1(e)$, $\mathrm{A}_2[h_2(e)]\times g_2(e)$, ... , $\mathrm{A}_d[h_d(e)]\times g_d(e)$. Because the reported value is a '`median`', the C sketch suffers from both over-estimation and under-estimation errors.

Based on the C sketch, the CM sketch [27] eliminates the under-estimation error. A CM sketch also consists of $d * w$ counters. When inserting an item e, it increments the d counters $\mathrm{A}_i[h_i(e)]$ ($1 \leqslant i \leqslant d$, we call them d `hashed counters` for convenience) by 1. The deletion is just the reverse of insertion. When querying an item e, it reports the minimum one among the d hashed counters $\mathrm{A}_i[h_i(e)]$ ($1 \leqslant i \leqslant d$). CU sketches [24] are very similar to CM sketches except the insertion: when inserting an item, CU sketches increase only the minimum counter(s) rather than all the d hashed counters. CU sketches work better than other sketches on three important NLP problems [29]. Although this improvement increases the query accuracy, it causes CU sketches do not support deletions. CML sketches [28] use logarithm-based approximate counters rather than linear counters. It increases the d hashed counters with logarithmic probabilities. This strategy makes such structures to record more (larger) frequencies with fewer bits for each counter but sacrificing the ability of supporting deletions. A very recent work called Augmented sketch [7] focuses on the accuracy optimization on hot items, and builds a filter (a queue) to dynamically store them, and use a sketch to store all item frequencies. In this way, Augmented sketch can improve the accuracy at the cost of complicated implementation and frequent communications between the filter and the sketch. There are two other typical sketches called Counter Braids [30] and random counters [31]. Counter braids sketch has a high accuracy when the memory usage is large. However, it cannot support instant query, which means that the query operation can only be performed after inserting all items, and its update and query speed are both slow. What is worse, when the memory size is small, its error rate of decoding drastically increases. To address these shortcomings, the random counter sketch only increases exact one counter for each update, thus is the fastest sketch among prior art. However, the accuracy of random counter sketch is not as good as that of the CU sketch.

Unfortunately, all these existing sketches do not address the problem of memory inefficiency. This problem drastically deteriorates for non-uniform datasets. To address this problem, we propose a novel sketch framework in the next section, enabling existing sketches work well for non-uniform datasets.

III. The ABC Framework

In this section, we present the details of our Adjacent Borrow and Carry (ABC) sketch framework which can be applied to all existing sketches. We first present the rationale of our ABC framework, and then present the two techniques: *bits-borrowing* and *combination* in details. Table I summarizes the symbols and abbreviations used in this paper.

A. Rationale

Sketches are intended for applications that do not require absolute accuracy but make use of estimations, even with large errors for a small portion of the estimations, as long as the overall statistical distribution stays about the same. Existing sketches hash each data item to multiple counters so that the probability of having a counter without collision is increased. In this way, as an example, the CM sketch will return d counter values for each item. As long as one of the d counters is collision-free, we will have an accurate answer. Even when all d counters contain errors due to collisions, the smallest counter has the least error. On top of multi-hashing, we can further improve the accuracy by increasing the number of counters, which reduces the chance of hash collisions and improves the likelihood for each item to have a collision-free counter. For a given amount of memory, more counters mean fewer bits per counter.

In practice, many multisets are non-uniform [7], [24]. What is worse, some multisets are extremely non-uniform. Specifically, the frequencies of most items in these multisets are small while only a small number of them are very large. On the one hand, in order to prevent counters from being overflowed by large-frequency items, we should adopt a large counter size, which means fewer counters and more collisions, causing over-estimation errors. Moreover, the dominance of low-frequency items means that the high-order bits of most counters are likely to be left unused, causing space waste. On the other hand, if we use small counters, there is the problem of overflow, resulting in exceptions and under-estimation errors.

Our quest is to solve the above dilemma faced by conventional sketches. The proposed ABC framework will use a large number of small counters, yet allowing overflowed counters to borrow space from other counters dynamically. There is a surprisingly rich design space with a variety of possible methods for storing overflowed counts in adjacent counters, combining adjacent counters to create larger counters, borrowing bits from other counters, or applying different methods jointly. The ABC framework will apply these methods to improve the accuracy of sketches as high as possible.

The major advantage of the ABC framework is that it solves the overflow problem such that we can use a large number of small counters to reduce collision-induced errors. We stress that the ABC framework does not require an error-free counter design. Instead, we will identify the sources of errors and try every means to reduce them in a series of designs. For the remaining errors in counters, as we apply the ABC framework to existing sketches, their aforementioned multi-hashing strategy will further reduce or sometimes eliminate the errors.

B. Technique I: bits-borrowing

Without loss of generality, a sketch is a matrix consisting of d arrays and each array consists of w counters. We represent the i^{th} array in the sketch with A_i, and the j^{th} counter of the i^{th} array in the sketch with $\mathrm{A}_i[j]$, where $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant w$. Each array A_i is associated with a uniformly distributed independent hash function $h_i(.)$ $(1 \leqslant h_i(.) \leqslant w)$.

TABLE I
SYMBOLS & ABBREVIATIONS USED IN THE PAPER

Symbol	Description
d	# of arrays of a sketch
w	# of counters in an array
$\overline{w}$	# of bits in an array
b	# of normal bits in a counter
e	one element
$\mathrm{A}_i[j]$	the counter in the j^{th} position of the i^{th} array
$\mathrm{A}_i[j]$.value	the value of the counter $\mathrm{A}_i[j]$
$\mathrm{A}_i[j]$.FG	the flag bit of the counter $\mathrm{A}_i[j]$
$\mathrm{A}_i[j]$.BW	the borrow bit of the counter $\mathrm{A}_i[j]$
$h_i(.)$	the i^{th} hash function
ABC	the Adjacent Borrow and Carry framework
Mqps	mega-queries per second

In our ABC framework, each counter has $b+1$ bits, of which b bits serve as a counter, recording how many insertions have occurred in this counter, and the rest 1 bit serves as a FlaG bit (FG). For definiteness and without loss of generality, we always choose the most significant bit of the counter as FG. The values of all counters $\mathrm{A}_i[j]$.value are initially 0, where $1 \leqslant i \leqslant d$ and $1 \leqslant j \leqslant w$.

In this part, we present our first technique: `bits-borrowing`. Using this technique, when counter overflow occurs, the overflowed counter will only *borrow free bits* from the next counter. Here free bits refer to the prefix 0s of the next counter. In this way, memory efficiency is improved.

Data Structure: As shown in Figure 1, each counter in our ABC framework has a flag bit (FG), and each *borrowed* counter is divided into two parts: the *borrowed space* and the *used space*. The borrowed space is borrowed by the previous counter, and the used space still belongs to the original counter. Each time when a counter (Counter A in the figure) overflows, it sets the flag bit to 1, sets the value of the counter to 0, borrows one free bit from the used space in the next counter, put it in the borrowed space, and sets the new added bit in the borrowed space to 1. For example, as shown in Figure 1, the *borrowed space* (green color) in Counter B is borrowed by Counter A while the *used space* (blue color) still belongs to Counter B, and there are three prefix 1s in the borrowed space, which indicates that Counter A has overflowed for three times. Note that there is always a 0-bit (we call it the *wall* for convenience) in the borrowed counter (Counter B), which is used to separate the borrowed space and the used space.

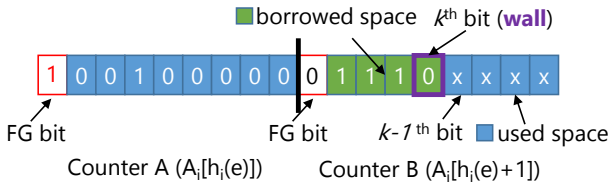

Fig. 1. The ABC framework using bits-borrowing technique.

Insertion: Using the bits-borrowing technique, there are three cases of insertions. Without loss of generality, we discuss insertions of the three cases only in the i^{th} array of the sketch.
Case 1: When inserting an item e, if $A_i[h_i(e)]$ does not overflow before and after the insertion, our framework does not modify the insertion process no matter $A_i[h_i(e)]$.FG is 0 or not.
Case 2: When inserting an item e, if $A_i[h_i(e)]$.FG is 1 (its counter has been overflowed before) and the counter is going to overflow again, we perform several operations on the `next` counter as follows. We search for the left-most '0' bit in the next counter: for example in Figure 1, the left-most '0' is the k^{th} bit. Then we check the $k-1^{th}$ bit. 1) If the $k-1^{th}$ bit is 1, which means it is a part of the recorded value of the borrowed counter, then the k^{th} bit should not be modified, because the 0 in the k^{th} bit is a necessary `wall` to separate the `borrowed space` and the `used space`. Therefore, there is no free bit to be borrowed, and another technique is required, which will be discussed in the next section. 2) If the $k-1^{th}$ bit is 0, then we turn the k^{th} bit from 0 to 1 and reset $A_i[h_i(e)]$.value to indicate that the previous counter overflows again.
Case 3: When the insertion of an item e is going to cause $A_i[h_i(e)]$ to overflow for the first time, we first check the two most significant bits of the next counter. If both of them are 0, we set $A_i[h_i(e)]$.FG to 1 and set the most significant bit (except for the flag FG) of $A_i[h_i(e)+1]$ to 1 to indicate $A_i[h_i(e)]$ overflows once. Otherwise, we do nothing because the `wall` to separate the borrowed space and the used space cannot be established, and this suggests that we need to borrow bits from the next adjacent counters using the same method.
Query: When querying the value of item e in the i^{th} array, we first check $A_i[h_i(e)]$.FG. If it is 1, then we count the number of 1s in the borrowed space of $A_i[h_i(e)+1]$. Suppose there are t successive 1s, then the query value of e is decoded as $A_i[h_i(e)]$.value$+t*2^b$. If $A_i[h_i(e)]$.FG is 0 but $A_i[h_i(e)-1]$.FG is 1, then $A_i[h_i(e)]$ is borrowed by the previous counter. Therefore, we ignore the successive 1s in the borrowed space of $A_i[h_i(e)]$, and the rest bits are decoded as the value of this counter. If both $A_i[h_i(e)]$.FG and $A_i[h_i(e)-1]$.FG are 0, then we report $A_i[h_i(e)]$.value as the query value.
Deletion: Deletion is exactly the inverse process of insertion. Specifically, when deleting an item e, if $A_i[h_i(e)]$.value is not 0, we simply decrease $A_i[h_i(e)]$.value. Otherwise, 1) if $A_i[h_i(e)]$.FG is 1, we will set $A_i[h_i(e)]$.value to the maximum value, find the least significant '1' bit in the borrowed space of $A_i[h_i(e)+1]$ and set the bit to 0; 2) if $A_i[h_i(e)]$.FG is 0, this means "deleting a nonexistent item", thus the deletion will be aborted.
Advantages and Limitations: The bits-borrowing technique shows a method to borrow free space from the next counter. This method significantly diminishes interaction between the overflowed counter and the borrowed counter. The updating of the overflowed counter would not `rob` the used bits of the borrowed counter and thus would not result in any deviation, at the cost of making the capacity of the borrowed counter smaller. At the same time, the updating of the borrowed counter would not influence the borrowed bits, and thus has no side effect on the accuracy of the overflowed counter. The main shortcoming of the bits-borrowing technique is that the allowed times of overflows of a single counter are limited. When there is no free bit in the next counter, the bits-borrowing technique cannot work, and another solution is required. In the next part, we present another technique to address this issue.

C. Technique II: combination

In this section, we propose another key technique used in our ABC framework, namely `combination`. *Its key idea is to combine two adjacent counters into one big counter when no free bit is available in the borrowed counter.*

For convenience, we name the second significant bit (the b^{th} bit) of the counter BorroW bit (BW). BW is used as an essential signal of whether the counter is being borrowed or being combined. When a counter overflows for the first time, our framework starts using the `bits-borrowing` technique. When Counter A overflows for $b-k$ times but the $k-1^{th}$ bit in Counter B is 1, as shown in Figure 2, Counter A is not capable to borrow more bits from Counter B. In this case, we use the `combination` technique and set BW in Counter B to 0 to indicate it is combined with Counter A as shown in Figure 3. Without loss of generality, we let bits in the left counter (Counter A) be the higher bits and let those in the right counter (Counter B) be the lower bits. Note that in Figure 3, BW does not belong to the big counter, because it should be always 0 when the two adjacent counters are combined.
Different combination policies for different sketches: When combining two adjacent counters in the sketch, we should let the big counter store the frequencies of both two adjacent counters. Note that for each incoming item, the CM sketch increments all mapped counters by 1, while the CU sketch only increments the counter(s) with the minimum value. Therefore, we should take different combination policies for different sketches. For CM sketches, we should set the value in the big counter to the sum of values in original adjacent counters, while for CU sketches, the value in the big counter should be set to the maximum of values in original counters. Table II shows different combination results of the big counter in different sketches.

TABLE II
DIFFERENT COMBINATION RESULTS OF THE BIG COUNTER IN DIFFERENT SKETCHES (c_1 AND c_2 ARE ORIGINAL VALUES IN THE TWO ADJACENT COUNTERS, κ IS THE LOG BASE OF CML SKETCHES)

Sketches	Combination result
CM sketches [27]	c_1+c_2
CU sketches [24]	$max(c_1,c_2)$
C sketches [26]	c_1+c_2
CML sketches [28]	$log_\kappa \frac{2-\kappa^{c_1}-\kappa^{c_2}}{1-\kappa}$

Next, we describe the method of insertion, deletion, and query in our ABC framework, applying both *bits-borrowing* and *combination* technique.

Insertion: When inserting an item e, we compute the hash functions $h_1(e)$, $h_2(e)$, ..., $h_d(e)$ and insert e to each array, respectively. Without loss of generality, we only discuss the insertion in the i^{th} array.

Case 1: We just increase $A_i[h_i(e)]$.value in the following three situations: 1) Both $A_i[h_i(e)-1]$.FG and $A_i[h_i(e)]$.FG are 0, and $A_i[h_i(e)]$ is not going to overflow; 2) $A_i[h_i(e)-1]$.FG is 0 and both $A_i[h_i(e)]$.FG and $A_i[h_i(e)+1]$.BW are 1, and $A_i[h_i(e)]$ is not going to overflow; 3) Both $A_i[h_i(e)-1]$.FG and $A_i[h_i(e)]$.BW are 1, and $A_i[h_i(e)]$ is not going to overflow.

Case 2: There are two situations: 1) Both $A_i[h_i(e)-1]$.FG and $A_i[h_i(e)]$.FG are 0, and $A_i[h_i(e)]$ is going to overflow; 2) $A_i[h_i(e)-1]$.FG is 0, both $A_i[h_i(e)]$.FG and $A_i[h_i(e)+1]$.BW are 1, and $A_i[h_i(e)]$ is going to overflow. In these two situations, if there is a bit available for borrowing in $A_i[h_i(e)+1]$, we will turn the bit into 1 and reset $A_i[h_i(e)]$.value. Otherwise, we will set $A_i[h_i(e)+1]$.BW to 0, which means combining $A_i[h_i(e)]$ with $A_i[h_i(e)+1]$ into a big counter.

Case 3: Both $A_i[h_i(e)-1]$.FG and $A_i[h_i(e)]$.BW are 1, and the used space of $A_i[h_i(e)]$ is going to overflow. We will set $A_i[h_i(e)]$.BW to 0, which means combining $A_i[h_i(e)]$ with $A_i[h_i(e)-1]$ into a big counter.

Case 4: We just increase the big counter in the following two situations: 1) $A_i[h_i(e)-1]$.FG is 0 and $A_i[h_i(e)]$.FG is 1 and $A_i[h_i(e)+1]$.BW is 0; 2) $A_i[h_i(e)-1]$.FG is 1 and $A_i[h_i(e)]$.BW is 0.

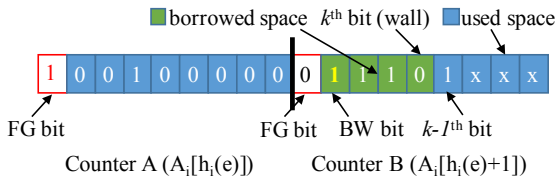

Fig. 2. The ABC framework - bits-borrowing

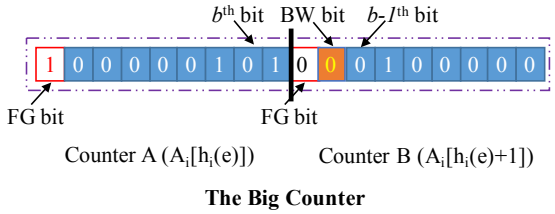

Fig. 3. The ABC framework - combination

Query: When querying an item e, we check all d arrays and each array will report a value, here we focus on how to report the i^{th} value.

Case 1: Both $A_i[h_i(e)-1]$.FG and $A_i[h_i(e)]$.FG are 0. Then we just report $A_i[h_i(e)]$.value as the i^{th} value.

Case 2: $A_i[h_i(e)-1]$.FG is 1, and $A_i[h_i(e)]$.BW is also 1. Then we check $A_i[h_i(e)]$ starting from the BW bit to locate the first '0' bit (wall), and report the rest bits on the right of the wall as the query value.

Case 3: $A_i[h_i(e)]$.FG is 1, and $A_i[h_i(e)+1]$.BW is also 1. First we get t, the number of borrowed bits in $A_i[h_i(e)+1]$, by counting the number of successive 1s down from the BW bit. Then we report $A_i[h_i(e)]$.value$+t\times 2^b$.

Case 4: $A_i[h_i(e)]$.FG is 1 and $A_i[h_i(e)+1]$.BW is 0, or $A_i[h_i(e)-1]$.FG is 1 and $A_i[h_i(e)]$.BW is 0. Then this is a big counter which has $2b-1$ bits. Note that the left counter represents the higher bits while the right counter represents the lower bits, we report $2^b\times A_i[h_i(e)-1]$.value$+A_i[h_i(e)]$.value in the former case, and $2^b\times A_i[h_i(e)]$.value$+A_i[h_i(e)+1]$.value in the latter case.

Deletion: We focus on the deletion in the i^{th} array.

Case 1: $A_i[h_i(e)]$ is not in a big counter and is not going to `underflow`, *i.e.*, its current value is not equal to 0. We simply decrease it.

Case 2: $A_i[h_i(e)]$ has borrowed bits from $A_i[h_i(e)+1]$ and is going to underflow. We reset the least significant bit of the borrowed space in $A_i[h_i(e)+1]$ and set $A_i[h_i(e)]$.value to the maximum value.

Case 3: $A_i[h_i(e)]$ is in a big counter. We just decrease the big counter.

IV. Mathematical Analysis

Our ABC framework can enable most existing sketches to work well for non-uniform datasets. In this section, we make mathematical analysis for CM sketches as the case study, and we call the CM sketch after using our ABC framework the CM-ABC sketch for convenience.

A. Proof of No Under-estimation Error

In this section, we prove there is no under-estimation error in the CM-ABC sketch. Under-estimation error means that *the querying value is smaller than the real frequency*. We claim that our CM-ABC sketch is free of under-estimation error, because every mapped counter maintains a larger value than the real frequency.

Theorem 1. *Our CM-ABC sketch is free of under-estimation error, because every mapped counter maintains a larger value than the real frequency.*

Proof. There are four cases for insertions.

Case 1: The insertion processes of CM-ABC and CM sketches are exactly the same when no overflows occur. Each mapped counter will be increased by 1 when inserting item e, so the estimated value must be no less than the real frequency of e.

Case 2: If a counter has borrowed bits from its next counter, the values of both counters are larger than the real frequencies of their corresponding items, since they do not influence each other.

Case 3: If a counter has been combined with the adjacent counter, the initial value of the big counter is the sum of the two counters. Therefore, the value of the big counter is larger than the real frequencies of all items mapped into these two counters.

Case 4: After insertions in a big counter, the value of the big counter is clearly larger than the real frequencies of all items mapped to this big counter.

As deletion is the inverse operation of insertion, the detailed derivation is ignored due to space limitation. According to the above derivation, it can be possible to conclude that the CM-ABC sketch has no under-estimation error.

□

B. Correct Rate of CM-ABC Sketches

Given a multiset, we build a CM-ABC sketch. Let n be the number of distinct items. Let w' be the number of counters in each array. Let σ be the average number of big counters in each array. Let d be the number of arrays in the CM-ABC sketch. Let C_r be the correct rate (the ratio of items whose frequency is equal to its estimate result to total items) of the CM-ABC sketch.

Theorem 2. *Given the parameters w', n, σ, and d as defined above, the correct rate C_r of the CM-ABC sketch is calculated by the following formula:*

$$C_r \approx 1 - \left(1 - \left(\frac{w'-1}{w'}\right)^{n-1} + \frac{2*\sigma}{w'}\right)^d \tag{1}$$

Proof. First, we suppose there are no overflows and thus no big counters in the sketch. For an arbitrary item e, in the i^{th} array the hashed counter stays the accurate value if and only if there is no collision in the counter, and the probability P is:

$$P = \left(\frac{w'-1}{w'}\right)^{n-1} \tag{2}$$

The item e suffers from a false positive only in the condition that there are collisions in all the d hashed counters. In this case, the query result of e returns an over-estimated value with the probability P':

$$P' = (1-P)^d = \left(1 - \left(\frac{w'-1}{w'}\right)^{n-1}\right)^d \tag{3}$$

Now, we consider the situation that counters overflow in the CM-ABC sketch. When a counter is borrowing bits from the adjacent counter, no positive false is introduced. However, after this counter is combined with the adjacent counter, items mapped into both two counters will suffer from false positives even if there are no other items hashed into the big counter, because the value of the big counter is larger than the value of the two small counters. As a result, a big counter introduces additional false positives. The number of big counters is denoted by σ. σ is determined by the distribution of the items, and can be easily obtained during insertions. Here σ should be doubled because two counters are combined into one big counter. Therefore, the ultimate correct rate formula is as follows:

$$\begin{aligned} C_r &\approx 1 - \left(1 - P\left(1 - \frac{2*\sigma}{w'}\right)\right)^d \\ &= 1 - \left(1 - \left(\frac{w'-1}{w'}\right)^{n-1}\left(1 - \frac{2*\sigma}{w'}\right)\right)^d \end{aligned}$$

□

Here we made a comparison between the correct rate of the CM sketch and the CM-ABC sketch. The correct rate of the CM sketch can be easily calculated:

$$C_r = 1 - \left(1 - \left(\frac{w-1}{w}\right)^{n-1}\right)^d$$

Therefore, if we set $b = 16$ for the CM sketch, and set $b = 8$ bits for the CM-ABC sketch, the width of the CM-ABC sketch w' is the twice as large as that of the CM sketch w. With fixed $n = 1000000$, $d = 4$ and memory size, we can get the correct rate of both two sketches. Here b is the counter size, d is the depth of sketches, w and w' is the width of sketches. As shown in Figure 4, with memory size varied from 1MB to 10MB, the correct rate of the CM-ABC sketch always has a great advantage over that of the CM sketch.

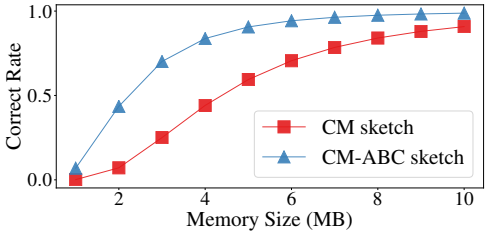

Fig. 4. Correct rate of CM-ABC and CM sketches.

C. Error Bound of CM-ABC Sketches

Theorem 3. *For each item x, we can ensure the probability that its query result v'_x exceeds its real frequency v_x by ϵ is bounded.*

$$Pr(v'_x > v_x + \epsilon \times V) \leqslant \left(\frac{w' + 4*\sigma}{\epsilon w'^2}\right)^d \tag{4}$$

Proof. We define $c_{x,j,y}$ as a boolean variable indicating whether item x is hashed into the same counter (including combined big counters) with as item y in the j^{th} array. The expectation of $c_{x,j,y}$ can be easily obtained:

$$E(c_{x,j,y}) = \frac{1}{w'} + \frac{2}{w'} \cdot \frac{2*\sigma}{w'} \tag{5}$$

We define $C_{x,j} = \sum_{y=1}^{n} c_{x,j,y} v_y$. Since hash functions are independent, we can get the expectation of $c_{x,j,y}$ as follows:

$$\begin{aligned} E(C_{x,j}) = E(\sum_{y=1}^{n} c_{x,j,y} v_y) &\leqslant V \sum_{y=1}^{n} E(c_{x,j,y}) \\ &\leqslant V \times \left(\frac{1}{w'} + \frac{2}{w'} \cdot \frac{2*\sigma}{w'}\right) \end{aligned} \tag{6}$$

By the Markov inequality,

$$
\begin{aligned}
Pr(v'_x > v_x + \epsilon \times V) &= Pr(\forall j, count[h_j(x)] > v_x + \epsilon \times V) \\
&= Pr(\forall j, v_x + C_{x,j} > v_x + \epsilon \times V) \\
&\leqslant Pr(\forall j, C_{x,j} > \epsilon \times V) \\
&\leqslant \left(\frac{E(C_{x,j})}{\epsilon \times V}\right)^d \leqslant \left(\frac{w' + 4 * \sigma}{\epsilon w'^2}\right)^d
\end{aligned} \tag{7}
$$

□

It is proved that for CM sketches, the error bound formula is

$$Pr(v'_x > v_x + \epsilon \times V) \leqslant \left(\frac{1}{\epsilon w}\right)^d$$

Clearly, if we use a counter size of 16 bits for the CM sketch, and use a counter size of 8 bits for the CM-ABC sketch, the CM-ABC sketch has a better error bound than the CM sketch. Figure 5 shows the different error bounds with different memory size for both two sketches ($\epsilon = 0.0001$).

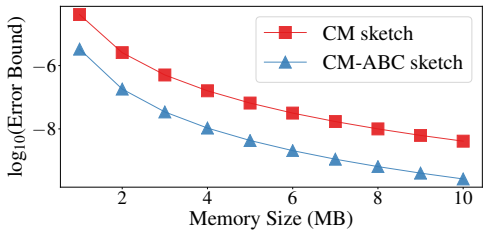

Fig. 5. Logarithm of error bounds of CM-ABC and CM sketches.

V. Experimental Results

In this section, we will present the experimental results of our framework when applied to the CM sketch and the CU sketch. There are two essential factors to evaluate: accuracy and query speed. We use the CPU platform with single core to evaluate the accuracy of our framework, and the multi-core CPU platforms to evaluate the query speed. The CM sketch is the most widely used sketch, and the CU sketch is the most accurate sketch [29]. We have made comparisons between the sketches (CM and CU sketches) before and after using the ABC framework.

A. Experimental Setup

Platform: Our platform is built on a machine with 12-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62 GB total DRAM memory. CPU has three levels of cache memory: two 32KB (where 1KB = 2^{10} bytes) L1 caches (one is a data cache and the other is an instruction cache) for each core, one 256KB L2 cache for each core, and one 15MB (where 1MB = 2^{20} bytes) L3 cache shared by all cores.

Real Datasets: We capture real IP packets from the main gateway of our campus. Sketches can be used to record the number of packets of each flow. One flow is often identified by the five-tuple: source IP address, destination IP address, source port, destination port, and protocol type. We use 100 groups of datasets, and each dataset has 10M packets. We show the average results of the 100 groups in the following experiments. According to our statistical results, the distribution of flows in these datasets is similar. Here we show the distribution of a typical dataset as follows. 1) There are around 300K distinct flows in each dataset. 2) The average frequency of all items is 33.92 and the deviation is 1095^2. 3) In the 300K distinct flows, there are about 173.5K items whose frequencies are 1, and there are 156 items whose frequencies are larger than 10000. For an item e with a frequency of 1, if it collides with a hot item with a frequency of 10,000, the estimated value of e will be 10,001, then the relative error is 10,000. That is why the following experimental results of relative error in average seem surprisingly large. Actually, even if the relative error is 1 or 2, the sketch is actually very accurate.

Uniform Datasets: we also conduct experiments on uniform datasets. As we did not find uniform datasets, we use the well known tool YCSB [32] to produce datasets in which the item frequencies follow uniform distribution. Each uniform dataset consists of 10M items, among which there are 1M distinct items. We repeat our experiments by generating multiple datasets, and the experimental results are almost the same.

B. Metrics

Relative Error (RE): RE(e) is defined as:

$$RE(e) = \frac{|\hat{f}(e) - f(e)|}{f(e)}$$

where RE(e) represents the relative error of the item e, $\hat{f}(e)$ represents the frequency reported by the sketch, and $f(e)$ represents the real frequency.

Average Relative Error (ARE): ARE is defined as:

$$ARE = \frac{1}{N}\sum_{i=1}^{N}\frac{|\hat{f}(e_i) - f(e_i)|}{f(e_i)}$$

where N is the number of items in the query set, $\hat{f}(e)$ represents the frequency reported by the sketch, and $f(e)$ represents the real frequency.

Throughput: The query speed is another important factor to evaluate. Here we use the throughput to measure the query speeds of those sketches.

C. Applied to the CM Sketch

In this part, we focus on the performance of the CM sketch before and after using our ABC framework, and we call the sketch after using the ABC framework CM-ABC sketch. To evaluate the accuracy of both sketches, we focus on the average relative error (ARE) of them after insertions and deletions. Because our ABC framework is tailored for non-uniform datasets, we will pay more attention to the experiments on IP packets which are non-uniformly distributed, while we also show that our framework does not bring more errors for synthetic uniform datasets. For experiments on real datasets, we set $d = 3$ and $\overline{w} = 1600000$; moreover, we set $b = 14$ for the CM sketch, and set $b = 7$ for the CM-ABC sketch. For experiments on uniform datasets, we set $d = 3$, $\overline{w} = 2000000$, and $b = 10$ for all sketches. Note that it is guaranteed that the memory usage is the same for all sketches.

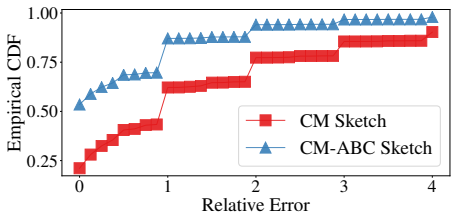

Fig. 6. CDF of relative error (Real datasets)

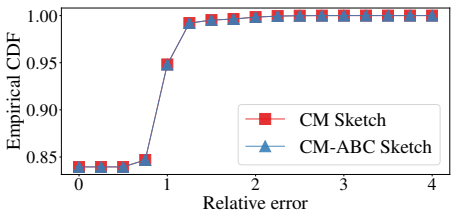

Fig. 7. CDF of relative error (Uniform datasets)

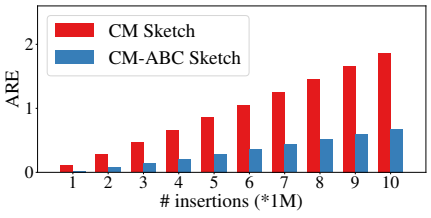

Fig. 8. RE vs. insertions (Real datasets)

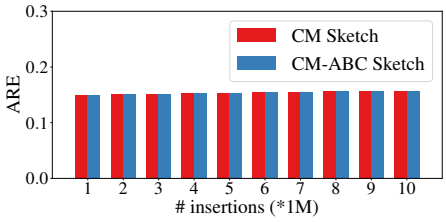

Fig. 9. RE vs. insertions (Uniform datasets)

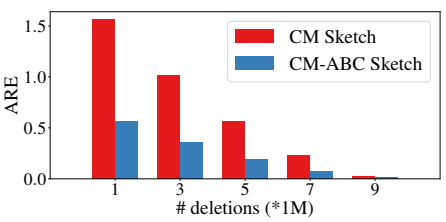

Fig. 10. RE vs. deletions (Real datasets)

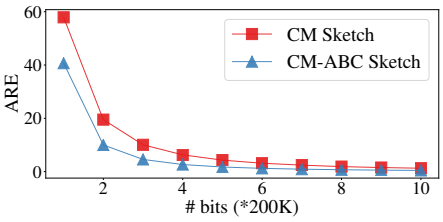

Fig. 11. Accuracy affected by $\overline{w}$

1) Relative Error and the Empirical CDF: For real datasets, our experimental results show that 53.49% items have no error for the CM-ABC sketch, while only 21.12% for the CM sketch. An intuition of the relative error can be made by Figure 6 which reveals the empirical CDF of both sketches. To be specific, we first performed a total of $10M$ insertions for $300K$ distinct items. After we completed these insertions, we calculated the relative error of each item, and thus we could compute the empirical CDF for both sketches and plot them. Just as shown in Figure 6, the CDF of the CM-ABC sketch is better than that of the CM sketch. In particular, for the ratio of those items that have no error, our ABC framework has a remarkable advantage. *For uniform datasets, our experimental results show that the CDFs of both sketches are very similar, and both sketches have 83.95% items that have no error.* Figure 7 shows the CDF of both sketches, and it shows that when working on synthetic uniform dataset, the accuracy of our CM-ABC sketch is as good as that of the CM sketch. In sum, after using the ABC framework, the accuracy of the CM sketch will increase significantly when applied to real non-uniform datasets, and keep unchanged when applied to uniform datasets.

2) Relative Error with Insertions: For real datasets, with different numbers of insertions spanned from $1M$ to $10M$, our experimental results show that the ARE of the CM-ABC sketch is $2.74 \sim 4.10$ (with a mean of 3.13) times smaller than that of the CM sketch. Figure 8 shows the ARE of both sketches with different numbers of insertions, and it shows that the CM-ABC sketch has a considerable advantage no matter how many insertions have made. *For uniform datasets, our experimental results show that the AREs of both sketches are very similar, with the ARE 0.157 for both sketches.* Figure 9 shows the ARE of both sketches with different numbers of insertions.

3) Relative Error with Deletions: For real datasets, our experimental results show that the ratio between the ARE of the CM sketch and that of the CM-ABC sketch ranges from 2.21 to 3.05 with different numbers of deletions ranging from $1M$ to $9M$. In this part, we first performed $10M$ insertions on both sketches, and then performed several distinct numbers of deletions. Last, we have plotted the ARE of those 100K distinct items in Figure 10. The experimental results of the uniform dataset are similar to those before. In other words, the performances of the two sketches are almost the same. Thus we *will not* present those results for following deletion experiments due to space limitation.

4) Effect of Sketch Parameters: In this part, we present the effect of sketch parameters including the width of the sketch and the number of arrays. We focus on the effect of the number of bits in each array ($\overline{w}$) and the number of arrays (d), thus we will fix other factors: $b = 7$ for sketches using the ABC framework and $b = 14$ for other sketches. In the experiment of varying $\overline{w}$, we make the sketches work on real datasets, and we vary $\overline{w}$ of both sketches from $400K$ to $2M$. In the experiment of varying d, we also make the sketches work on real datasets, and we vary d of both sketches from 2 to 10.

In the experiment of varying parameter $\overline{w}$, our experimental results show that the ARE of the CM-ABC sketch is always smaller than that of the CM sketch, and when $\overline{w}$ equals to 2M the ARE of the CM-ABC sketch is less than 0.43. And to be specific, we fix d at 4, like prior parts of the experiment. Figure 11 reveals the ARE with different width of all sketches, and it shows that as the width increases, the ARE declines steadily.

In the experiment of parameter d, our experimental results show that with a fixed width and different numbers of arrays, the ARE of the CM-ABC sketch is always smaller than that of the CM sketch. Figure 12 shows how the ARE changes of both sketches as d increases.

D. Applied to the CU Sketch

Now we evaluate the performance of the CU sketch before and after using the ABC framework, and we call the sketch after using the ABC framework CU-ABC sketch. And all the

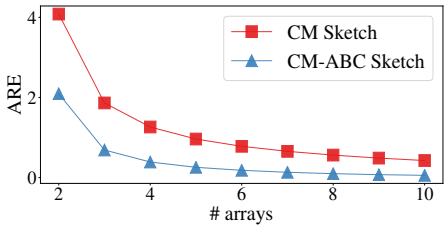

Fig. 12. Accuracy affected by d

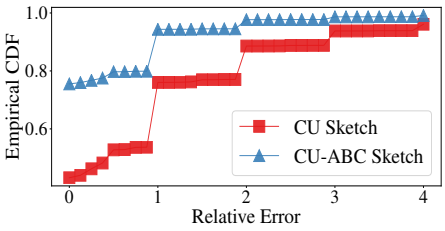

Fig. 13. CDF of relative error (Real datasets)

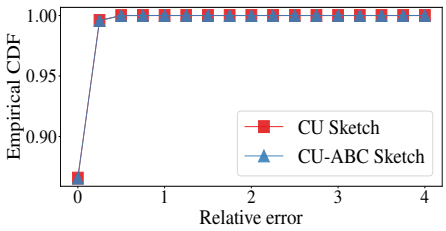

Fig. 14. CDF of relative error (Uniform datasets)

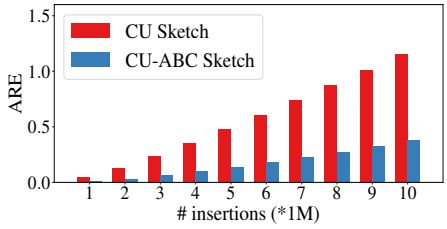

Fig. 15. RE vs. insertions (Real datasets)

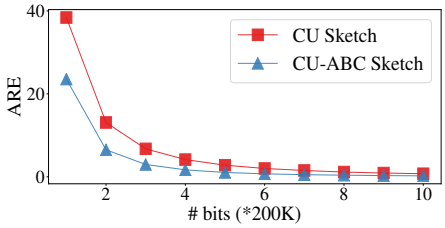

Fig. 16. Accuracy affected by $\overline{w}$

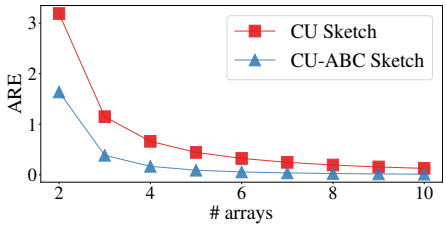

Fig. 17. Accuracy affected by d

parameter settings are the same as the previous parts. Due to space limitation, we only present the experimental results on real datasets. Furthermore, note that the CU sketch does *not* support deletion operation, thus we ignore the deletion experiments for the CU sketch.

1) Relative Error and the Empirical CDF: The experimental results show that our ABC framework works pretty well, and 75.52% items of the CU-ABC sketch have no error, while only 43.13% items of the CU sketch. Figure 13 reveals that the CDFs of both sketches, which indicates that our framework has a remarkable advantage.

For uniform datasets, our experimental results show that both sketches work well on this occasion, and no item suffers a relative error higher than 0.75. Figure 14 shows that more than 85% items suffer no error for both sketches and the difference of the performance of these sketches is negligible.

2) Relative Error with Insertions: Our experimental results show that the ARE of the CU-ABC sketch is $3.02 \sim 4.49$ *(with a mean of 3.47) times smaller than that of the CU sketch.* Figure 15 shows the ARE of both sketches with different numbers of insertions for real datasets, and it shows that the CU-ABC sketch has an advantage for different numbers of insertions.

3) Effect of Sketch Parameters: In the experiment of parameter $\overline{w}$*, our experimental results show that the accuracy of our CU-ABC sketch is much better than that of the CU sketch.* Figure 16 shows the result of this part of the experiment. As $\overline{w}$ increases, the ratio between the ARE of the CU-ABC sketch and that of the CU sketch declines to 0.3, *i.e.*, the accuracy of the CU-ABC sketch is at least 3 times better than that of the CU sketch.

In the experiment of parameter d*, our experimental results show that the ratio between the ARE of the CU sketch and the CU-ABC sketch ranges from 2.0 to 10.0.* Figure 17 shows the results of relative errors as d increases.

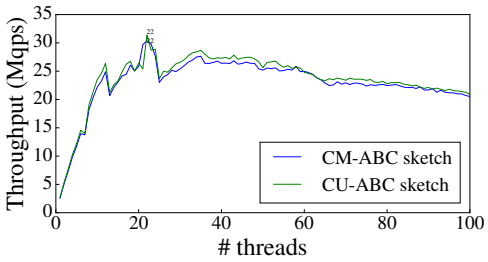

Fig. 18. Throughput vs. # threads

E. Speed Evaluation

In this section, we focus on the speed evaluation of the sketches using our ABC framework on multi-core CPU. Our experimental results on the multi-core CPU platform show that the query speed of CM and CU sketches are similar to those after using our ABC framework. Therefore, we only show the speed results of sketches after using our framework. We insert 10M items from real datasets and query 100K random items for these two sketches. Besides, we repeat each query operation 100 times and use the average value to minimize accidental errors.

Our experimental results show that the query speeds of both two sketches grow almost linearly as the number of threads increases. The query speeds of these two sketches can reach 30 Mqps (mega-queries per second). As shown in Figure 18, the query speeds of these two sketches achieve around 2.5Mqps for 100K queries with one thread, which achieves similar speed as the CM sketch and the CU sketch do. When using 22 threads, the query speeds of these two sketches reach the peak. This phenomenon can be explained by the fact that our

CPU has 2×6 cores with Hyper-Threading, which can handle $2 \times 6 \times 2 = 24$ concurrent threads, and there are some small deviations from the theoretical peak because the main thread responsible for detecting the query speed would occupy one or two cores. Furthermore, it can be inferred that the query speed of these two sketch could grow almost linearly with more CPU cores.

VI. Conclusion

Sketches have been applied to various fields and achieved great success. However, existing sketches can seldom handle the overflow problem of counters well, especially when sketches work for non-uniform datasets. As a result, existing sketches have to allocate more bits for each counter so as to alleviate the problem. However, this incurs much more memory usage but still cannot solve this problem in the worst case. To address this problem, in this paper, we propose a novel sketch framework - the ABC sketch framework, which exhibits good scalability and can be applied to most existing sketches. Using our ABC framework, sketches only use a few bits for each counter but achieve higher accuracy for non-uniform datasets. The key idea of our ABC framework is to borrow space from the adjacent counters by operations of bits-borrowing and combination when a counter is going to overflow. We applied the ABC framework to two typical sketches: the CM sketch and the CU sketch as case studies. We carried out extensive experiments, and experimental results show that the sketches after using the ABC framework significantly outperform those that not using the framework. We believe that our ABC framework can be applied to many more sketches and other similar data structures.

References

[1] "Source code of ABC models." http://net.pku.edu.cn/~yangtong/pages/ABC.html.

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

[3] G. Cormode and M. Garofalakis, "Sketching streams through the net: Distributed approximate query tracking," in *Proc. VLDB*, 2005.

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

[5] A. Goyal, H. Daumé III, and S. Venkatasubramanian, "Streaming for large scale nlp: Language modeling," in *Proceedings of Human Language Technologies*. Association for Computational Linguistics, 2009.

[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."

[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] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: A sketch framework for frequency estimation of data streams," *Proceedings of the VLDB Endowment*, vol. 10, no. 11, pp. 1442–1453, 2017.

[10] N. Polyzotis, M. Garofalakis, and Y. Ioannidis, "Approximate xml query answers," in *Proc. ACM SIGMOD*, 2004.

[11] J. Spiegel and N. Polyzotis, "Graph-based synopses for relational selectivity estimation," in *Proc. ACM SIGMOD*, 2006.

[12] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2, 1996, pp. 103–114.

[13] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy, "Tracking join and self-join sizes in limited storage," in *Proc. ACM SIGMOD*, 1999, pp. 10–20.

[14] M. Thorup and Y. Zhang, "Tabulation based 4-universal hashing with applications to second moment estimation." in *SODA*, vol. 4, 2004, pp. 615–624.

[15] A. Pietracaprina, M. Riondato, E. Upfal, and F. Vandin, "Mining top-k frequent itemsets through progressive sampling," *Data Mining and Knowledge Discovery*, vol. 21, no. 2, pp. 310–326, 2010.

[16] N. Pham and R. Pagh, "A near-linear time approximation algorithm for angle-based outlier detection in high-dimensional data," in *Proc. ACM SIGKDD*, 2012.

[17] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *Proc. IEEE ICDE*, 2004.

[18] G. Kollios, J. W. Byers, and et al., "Robust aggregation in sensor networks." *IEEE Data Eng. Bull.*, vol. 28, no. 1, pp. 26–32, 2005.

[19] A. Kleerekoper, M. Luján, and G. Brown, "Exploring sketches for probability estimation with sublinear memory," in *IEEE International Conference on Big Data*, 2013.

[20] P. Zhao, C. C. Aggarwal, and M. Wang, "gsketch: on query estimation in graph streams," *Proc. VLDB*, 2011.

[21] C. C. Aggarwal and K. Subbian, "Event detection in social streams." in *SDM*, vol. 12. SIAM, 2012.

[22] T. Yang, A. X. Liu, M. Shahzad, D. Yang, Q. Fu, G. Xie, and X. Li, "A shifting framework for set queries," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3116–3131, 2017.

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

[24] G. Cormode, "Sketch techniques for approximate query processing," *Foundations and Trends in Sample. NOW publishers*, 2011.

[25] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," *Proc. SIGMOD 2018*.

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

[27] 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.

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

[29] A. Goyal, Daume, H. Iii, and G. Cormode, "Sketch algorithms for estimating point queries in nlp," in *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

[30] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," *measurement and modeling of computer systems*, 2008.

[31] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEEACM Transactions on Networking*, 2012.

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