Compressed Sketches

Paper #530, 9+1 page

ABSTRACT

A sketch is a simple compact data structure for representing a data stream in order to support frequency estimation. Owing to its memory efficiency, it has gained wide acceptance, though it allows small error. In practice, sketches are often passed as messages. For example, in a data center, one often deploys many measurement nodes, each of which uses a sketch to record frequency information. To well understand the status of the whole data center, one often needs to collect all sketches periodically. In this case, the transmission size of the sketch is a limiting factor. In this paper, we proposed two algorithms to compress sketches, improving accuracy when the sketch is to transmit. Our compression method is applicable to most existing sketches. The cost is the processing time for compression. The compression time is short because of good cache behaviors. Different from normal compression methods, our compressed sketches do not need decompression. Related source code have been released on Github [1].

1 INTRODUCTION

1.1 Background and Motivation

Sketches are a kind of simple compact data structures for approximately recording data streams to support frequency estimation. Many well-known sketches have been proposed as described in Section 2.3, including sketches of CM [2], CU [3], Count [4], *etc.*The advantage of these sketches is that the insertion and query speed are fast and constant. The cost is that these sketches either have two-side error (both over-estimation error and under-estimation error), or one-side error (only over-estimation error). Fortunately, these errors are often small even using small memory, making sketches useful and popular.

Owing to the memory efficiency and fast processing speed, sketches have been applied to many fields. For example, in data stream scenarios, researchers have proposed many sketches for frequency estimation and heavy hitter detection, such as sketches of CM [2], CU [3],Count [4], CSM [5] and [6–8]. Researchers use sketches to detect DDoS attack [9, 10]. In NLP applications, sketches help analyze accurate statistics from large corpora [11–13]. For XML streams, sketches can be used to do summarization of ordered XML streams[14]. In database systems, sketches have been used as estimators to record frequency information [2, 4, 15–19]. And some recent work uses sketch variants to record persistence [20, 21] or existences [22, 23]. Especially, in many distributed systems, sketches are used to approximately record information in measurement nodes [24]. For example, Cormode *et al.* use sketches to continuously track a broad class of complex aggregate queries in a distributed-streams setting [25]. Andrea *et al.* use sketches to find global heavy hitters over distributed datasets [26]. Papapetrou *et al.* propose an ECM sketch to answer complex queries over distributed data streams in the sliding-window model [27]. In distributed systems, users often collect sketches from all measurement nodes periodically, and then do complex analyses and answer queries of the whole distributed systems. In these cases, sketches are passed as messages, and need bandwidth to send.

In distributed systems, the bandwidth is a key factor for two reasons: 1) The bandwidth is often a precious resource, especially in the mobile network. 2) The bandwidth often changes a lot and is hard to predict. Many literatures study the accuracy and speed improvement for sketches, while a very few works focus on the problem of bandwidth. In literature [21], the authors propose a method for merging sketches in middle nodes, so as to reduce the bandwidth usage. This solution is elegant and effective. A higher goal is to send a sketch with an appropriate size fitting to the current available bandwidth. Towards this goal, one straightforward solution is to predict the available bandwidth, and then determine the size of sketch $\mathcal{S}_0$ to build. However, the prediction is often very hard, and impossible to be exactly correct. Compared to such a prediction method of producing a sketch $\mathcal{S}_0$, this paper targets at the ideal goal: 1) When the sketch is to transmit, we can quickly generate a sketch $\mathcal{S}_1$ with an appropriate size without prediction. 2) We want to achieve that $\mathcal{S}_1$ has much higher accuracy than $\mathcal{S}_0$ when using the same size of bandwidth.

1.2 Our Solution

To compress a sketch, one may use classical compression methods, *e.g.*Huffman Coding [28], LZW [29], *etc.*However, these methods have to make trade-off between compression ratio and compression/decompression time. This paper finds another direction: we target at eliminating the decompression process and meanwhile achieving fast compression.

In this paper, we propose two compression algorithms of sketches, namely Adjacent Combining Compression (ACC$_1$ and ACC$_2$), achieving the above ideal goal. In addition, ACC$_1$ and ACC$_2$ are generic, fast and do not need decompression. Most existing sketches are composed of counters and hash functions. Our compression algorithms need to change both the hash functions and the value of counters. The main idea is combining every γ adjacent counters into one counter, where γ is the compression ratio. Next we show the key ideas of ACC$_1$ and ACC$_2$.

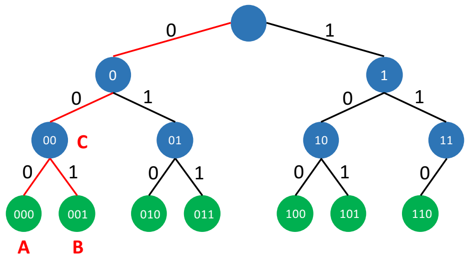


Figure 1: An example of trie.

The key idea of ACC$_1$ is called `Hash Shifting`, which is inspired by the observation of *trie* data structure. As shown in Figure 1, there is a trie which represents many sets. Each node in the tree represents a set, and its two child nodes represent its two subsets. The ID of each subset is the code of the path from the root node to the current node. For example, the IDs of node A and B are 000 and 001, respectively. If A and B are combined as node C, the new ID will be 00. After combination, the ID shifts one bit to the right, *i.e.*, the least significant bit is discarded. This regulation also fits into hash tables and sketches.

Inspired by this observation, given a sketch, ACC$_1$ first combines every $\gamma = 2^{\alpha}$ adjacent counters into one, and uses the sum of the γ counters as the new value. After compression, the hash functions become $h(.) \gg \alpha$ from $h(.)$, where $\gg$ means "right shift".

ACC$_2$ inherits the Hash Shifting technique, and uses another key technique named `Max Selection`. ACC$_2$ is only suitable for one-side error sketches defined in the first paragraph of this section. Max Selection is based on a `key observation`: given any adjacent γ counters and the maximum value among them, C_{max}, we observe that any item that mapped into these counters has a frequency less than or equal to C_{max}. Therefore, instead of setting the value of the compressed counter to the sum of the γ counters, Max Selection sets the value of the compressed counter to C_{max}. Theoretically, we have proved that 1) after compressing the sketches of CM and CU, the compressed sketches still keep the characteristic of one-side error; 2) for sketches of CM and CU, ACC$_2$ achieves higher accuracy than ACC$_1$.

1.3 Key Contributions

- We propose two generic compression algorithms to make sketches fitting to bandwidth without prediction, namely ACC$_1$ and ACC$_2$. They are fast and do not need decompression.
- We apply our compression algorithms to three typical sketches: CM, CU, and Count. We conduct some theoretical proofs, which validate that ACC$_1$ and ACC$_2$ are effective.
- Experimental results show that sketches using ACC algorithm achieves higher accuracy than strawman methods, and the cost is the compression time, which is very short because of good cache behaviors.

2 BACKGROUND AND RELATED WORK

In this section, we first describe the data stream model used in our paper. Then, we discuss the challenges of sending sketches with limited available bandwidth. Finally, we introduce three typical sketches.

2.1 Data Stream Model

In this paper, we use a simple and widely used *cash register* model of data streams. Assume that a data stream S has n items. It is denoted as $S = (a_1, a_2, \cdots, a_n)$, where the i-th item $a_i = e_j$ is one item in the item space $U = (e_1, e_2, \cdots, e_n)$. We use f_e to denote the frequency of item e in the data stream S. The items in the data stream can be identified by their item IDs. For example, the source IP address can be used as the item ID in IP traffic stream, the user ID can be used as the item ID in transaction stream, *etc.*

2.2 Challenges Of Adapting to Bandwidth

Nowadays, many systems are implemented in a distributed fashion, and multiple processors communicate with each other over a common network. In these distributed systems, the communication delay has a great effect on the system performance and can be the major problem especially in the systems which have many nodes and a high communication frequency. Therefore, the bandwidth for communication is a precious resource. However, network congestion is common in data centers. It may happen many times in a second [30], and be even as large as more than half of the whole bandwidth [31].

In data centers, sketches are often used to collect information for managing the state of the whole network. When network congestion happens, the available bandwidth left for sending sketches is small. If the sketch size is large, it can be delayed to send to the control server. On the one hand, because sketches carry important information of the state of the network, the network problem cannot be handled immediately. On the other hand, sending large size of sketches may affect the normal data traffic.

An ideal solution is to predict the available bandwidth in advance, and build a sketch with an appropriate size. However, the prediction is hard and hardly accurate. In this paper, we propose a compressing method for generating sketches with appropriate sizes without prediction.

2.3 Typical Sketches

Due to the significance, there are many works about sketches [23, 32, 33]. In this section, we mainly introduce typical sketches for frequency estimation.

Assume a sketch has d counter arrays, each of which has w counters. Let C_i be the i-th counter array in a sketch, $C_i[j]$ be the j-th counter in the i-th counter array, and $h_i(.)$ be the i-th hash function used in a sketch.

CM sketch [2] contains d counter arrays and d hash functions as explained in Figure 2. The counters are initialized to all zeroes. To record an item e, it calculates d hash values on e and locates d counters, $C_1[h_1(e)]$, $C_2[h_2(e)]$, . . ., $C_d[h_d(e)]$. Then, it increases these counters by 1. To query an item e, it calculates d hash values on e and reports the minimum value among them as the estimated frequency for e.

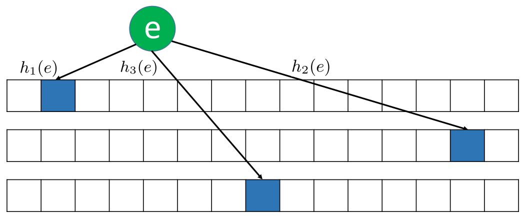


Figure 2: CM sketch. It contains three counter arrays and hash functions. To insert an item e, it locates three counters by calculating hash values on e, and increases them by 1. To query e, it locates the same counters as insertion, and reports the minimum value of them as the estimated frequency.

CU sketch [3] contains d counter arrays and d hash functions. The counters are initialized to all zeroes. To record an item e, it also locates d counters by hash computations. But it increases the smallest counter(s) of them by 1. To query an item e, it calculates d hash values on e, and reports the minimum value of $C_1[h_1(e)]$, $C_2[h_2(e)]$, . . ., $C_d[h_d(e)]$ as the estimated frequency for e.

Count sketch [4] contains d counter arrays and $2d$ hash functions. The counters are initialized to all zeroes. To record an item e, it firstly calculates $h_1(e), h_2(e), \ldots, h_d(e)$. Then, for each $C_i[h_i(e)], 1 \leqslant i \leqslant d$, it increases the counter by 1 or -1 decided by the parity of $h_{d+i}(e)$. To query an item e, it reports the median value of $C_1[h_1(e)], C_2[h_2(e)], \ldots, C_d[h_d(e)]$ as the estimated frequency for e.

3 COMPRESSED SKETCHES

In this section, we first introduce a strawman solution to make sketches adapted to the available bandwidth in Section 3.1. Then, we present two versions of our proposed algorithms to compress sketches: ACC$_1$ and ACC$_2$. Two key techniques include `Hash Shifting` in Section 3.2.1 and `Max Selection` in Section 3.2.2. Finally, we show an extension of our algorithm.

3.1 Strawman solution: Hierarchical Sketches

In order to adapt to the available bandwidth which changes drastically, the size of the sketch to send has to fit to the available bandwidth. As shown in Figure 3, a straightforward solution is to maintain multiple sketches (*e.g.*, multiple CM sketches) for one data stream, and choose one or more sketches with appropriate sizes to send according to the current available bandwidth. We call this strawman solution the *hierarchical sketches* for convenience. Below we describe the details of the hierarchical sketches.

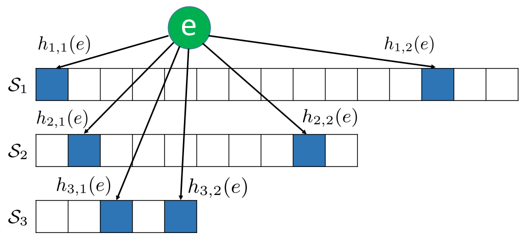


Figure 3: Hierarchical Sketch. It contains three CM sketches, each of which has two hash functions. To record an item e, these sketches calculate hash values on e and locate 6 (blue) counters, and increase them by 1.

Data structure: As shown in Figure 3, the hierarchical sketch contains multiple sketches with different sizes. Let $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_\rho$ be the multiple sketches (*e.g.*, CM sketches), where ρ is the number of sketches. Each sketch has d hash functions.

Insertion and Sending: To insert an item e, we insert e to all the ρ sketches. When we need to send a sketch, given an estimation of the current available bandwidth, we can choose a group of the multiple sketches according to following formula:

$$\{S_i \mid i \in \mathcal{I}_{\max}\}$$
$$\mathcal{I}_{\max} = \arg_{\mathcal{I}} \max \left\{ \sum_{i \in \mathcal{I}} s_i \mid \sum_{i \in \mathcal{I}} s_i \leqslant B, \mathcal{I} \subseteq \{1, \cdots, \rho\} \right\} \qquad (1)$$

where B is an appropriate size based on the estimation of available bandwidth, and s_i is the size of sketch $\mathcal{S}_i$ $(1 \leqslant i \leqslant \rho)$.

The reason for choosing multiple sketches instead of a single sketch to send is that, multiple sketches still make up a hierarchical sketch, so they can be reduced again when to be sent.

These hierarchical sketches can adapt to the available bandwidth. Obviously, multiple sketches require much more memory, and the insertion speed is low because of too many hash computations and memory accesses. An elegant solution is using only one sketch for recording, and compressing the sketch to an appropriate size when it needs to be sent. Next we show how to implement such a solution.

3.2 Compressed Sketches

In this section, we propose two compression algorithms, which compress the sketch when it is to be sent, improving both processing speed and accuracy. The main idea of our compression algorithms is combining adjacent counters into one counter, and thus we call our algorithms Adjacent Combining Compression (ACC). It has two versions: a basic version ACC$_1$ and an optimization version ACC$_2$.

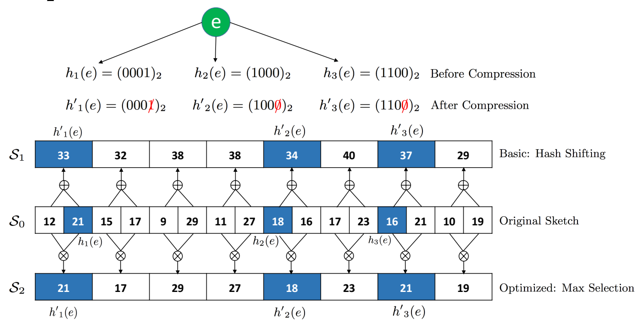


Figure 4: ACC$_1$ and ACC$_2$. $\mathcal{S}_0$ is the original sketch with 16 counters. $\mathcal{S}_1$ and $\mathcal{S}_2$ are the compressed sketches using ACC$_1$ and ACC$_2$, respectively. The compression ratio is two. $\oplus$ means add operation, and $\otimes$ means taking the largest one.

3.2.1 *ACC$_1$ Using Hash Shifting.*

The compression processing of ACC$_1$ is concise: given a sketch with $\gamma * w$ counters and hash functions, we divide it into w groups, each of which has γ counters. Every γ counters will be compressed into one counter, which is called the `compressed counter`. After grouping, there are two key issues: 1) How to choose the value of each compressed counter? 2) After compression, what are the new hash functions?

For the first issue, we propose a technique called `sum selection` to compute the value of the compressed counters. Often, the value of each counter in a sketch represents the sum of frequencies of all items that are mapped into this counter. Therefore, for any γ counters, we use the sum of them as the value of the compressed counter. This method is natural, and later it will exhibit good generality.

For the second issue, we propose the `hash shifting` technique. It works as follows. Let $\mathcal{S}[i\gamma]$, $\mathcal{S}[i\gamma + 1]$, ..., $\mathcal{S}[(i + 1)\gamma - 1]$ be the i^{th} group of counters in the original sketch S, and they will be compressed into the i^{th} counter $\mathcal{S}'[i]$ in the compressed sketch. Let $h_1(.), h_2(.), \ldots, h_d(.)$ be the original d hash function in sketch S. After compression, interestingly, these hash functions will become $h_1(.) \gg \alpha$, $h_2(.) \gg \alpha$, ..., $h_d(.) \gg \alpha$. The goal is using the old hash values (for determining the counters in $\mathcal{S}$) to determine the right counter in $\mathcal{S}'$ which can represent the old counters. From the observation that $(i\gamma) \gg \alpha = i, (i\gamma+1) \gg \alpha = i, \ldots, [(i + 1)\gamma - 1] > > \alpha = i$, the new hash functions become $h_1(.) \gg \alpha$, $h_2(.) \gg \alpha$, ..., $h_d(.) \gg \alpha$, where $\gamma = 2^\alpha$ and $\gg$ means right shift operation.

Next, we describe the compression and querying processing of our basic algorithm, ACC$_1$. Assume that the sketch to be compressed is $\mathcal{S}$ containing 2^s counters, and $\mathcal{S}[i]$ is the i-th counter in $\mathcal{S}$ ($0 \leqslant i < 2^s$). Let $h_1(.)$, $h_2(.)$, ..., $h_d(.)$ be the hash functions used in $\mathcal{S}$, where d is the number of hash functions. To determine counters for an item e in $\mathcal{S}$, we compute d s-bit hash values on e and locate d counters: $\mathcal{S}[h_1(e)]$, $\mathcal{S}[h_2(e)]$, ..., $\mathcal{S}[h_d(e)]$. Supposing the compression ratio is γ, when compressing, we first divide the counters in $\mathcal{S}$ into $\frac{2^s}{\gamma}$ groups. For the i-th group ($0 \leqslant i < \frac{2^s}{\gamma}$), it contains γ counters: $\mathcal{S}[i\gamma]$, $\mathcal{S}[i\gamma + 1]$, ..., $\mathcal{S}[(i + 1)\gamma - 1]$. For each group, we use the sum of the counters in it to form a new counter in the compressed sketch $\mathcal{S}_1$. We call this `sum selection` technique. When querying the frequency of e in $\mathcal{S}_1$, the hash functions to locate counters become $h_1(.) \gg \alpha$, $h_2(.) \gg \alpha$, ..., $h_d(.) \gg \alpha$, where $\gamma = 2^\alpha$ and $\gg$ means the right shift operation. We call this `hash shifting` technique.

For example, in Figure 4, we have a sketch $\mathcal{S}_0$ with 16 counters, and use three hash functions $h_1(.), h_2(.), h_3(.)$ for generating 4-bit hash values to locate counters in $\mathcal{S}_0$. When the compression ratio is two, for each two contiguous counters in $\mathcal{S}_0$, we add them up and form a new counter in $\mathcal{S}_1$. Before compression, we compute three hash values on e, $h_1(e) = (0001)_2$, $h_2(e) = (1000)_2$, $h_3(e) = (1100)_2$, and locate three (blue) counters in $\mathcal{S}_0$ whose values are 21, 18, 16, respectively. Thus, the estimated frequency given by $\mathcal{S}_0$ for e is $min\{21, 18, 16\} = 16$. After compression, we shift the three hash values right by one bit, $h'_1(e) = (000)_2$, $h'_2(e) = (100)_2$, $h'_3(e) = (110)_2$, and locate three (blue) counters in $\mathcal{S}_1$ whose values are 33, 34, 37, respectively. Thus, the estimated frequency given by $\mathcal{S}_1$ for e is $min\{33, 34, 37\} = 33$.

3.2.2 *ACC$_2$ with Max Selection.*

It is rarely the best choice to use the sum of the original γ adjacent counters as the value of the compressed counter. Take the CM sketch as an example. For each counter of a CM sketch, it stores the sum of the frequencies of all elements mapped to this counter. If we can build a hypothetical sketch S_h similar to CM sketch, except that each counter records the maximum frequency of the elements that mapped to this counter, then S_h can guarantee one-side errors, and achieves much higher accuracy than the CM sketch.

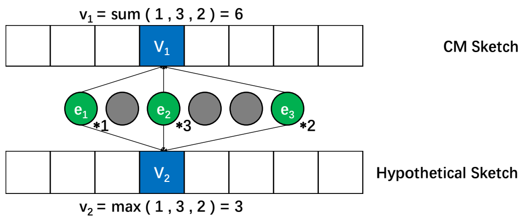


Figure 5: The CM sketch and the hypothetical sketch deal with the same data stream. e_1, e_2, e_3 are mapped to the same counter.

If the same element arrives in the data stream continuously, then we can successfully build the hypothetical sketch S_h. As shown in Figure 5, elements e_1, e_2, and e_3 are mapped to the same blue counter, and each element appears continuously 1, 3, and 2 times, respectively. The value of the counter is 0 at the beginning. When e_1 is inserted, the counter becomes 1; when e_2 is inserted, the counter becomes 3; when e_3 is inserted, the counter does not change. In the end, the value of the counter is $v_1 = \max\{1, 3, 2\} = 3$. If we construct a CM sketch, the corresponding counter of the CM sketch is $v_2 = \sum\{1, 3, 2\} = 6$, which is a serious over-estimation. Unfortunately, in practice, data streams seldom satisfy the "continuous arriving" requirement. Therefore, it is extremely difficult to build a sketch with the same accuracy as S_h. This paper is aimed at approaching the accuracy of S_h.

Back to our compression algorithm, in the method of ACC$_1$ mentioned above, we merged γ adjacent counters, and use the "sum selection". Based on the above analysis, we can use "max selection" instead of "sum selection". Specifically, we use the maximum counter of the original γ adjacent counters as the value of the compressed counter. Obviously, this optimization improves accuracy. Further, this optimization keeps one-side error for sketches of one-side error. The reason behind is that any item that was mapped into the γ adjacent counters has a frequency less than or equal to the maximum counter of the γ counters. One may want to use a smaller value rather than the maximum one. Unfortunately, if we use any value smaller than the maximum one, the characteristic of one-side error can not be guaranteed. For convenience, we refer to the compression algorithm using `Hash Shifting` and `Max Selection` as ACC$_2$.

We use an example to show that ACC$_2$ improves the accuracy of frequency estimation over ACC$_1$. in Figure 4, the estimated frequency given by $\mathcal{S}_0$ is $min\{21, 18, 16\} = 16$, and the estimated frequency given by $\mathcal{S}_1$ is $min\{33, 34, 37\} = 33$. The sketch using ACC$_1$ over-estimates the frequency of e almost twice. If we use the ACC$_2$ to compress $\mathcal{S}_0$ into $\mathcal{S}_2$ and the compression ratio is 2, for each two contiguous counters in $\mathcal{S}_0$, we select the larger one to form a new counter for $\mathcal{S}_2$. When querying the frequency of e, we locate three counters whose values are 21, 18 and 21, respectively. Thus, the estimated frequency given by $\mathcal{S}_2$ is $min\{21, 18, 21\} = 18$, which is more accurate than the estimated frequency given by $\mathcal{S}_1$ using ACC$_1$.

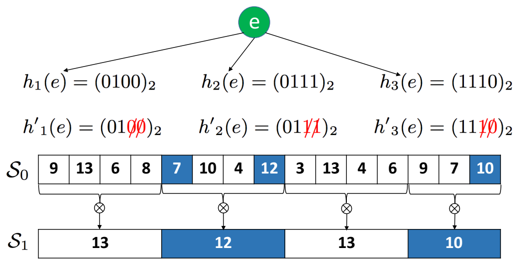


Figure 6: Our algorithm works when the original sketch size is not a power of 2. $\mathcal{S}_0$ is the original sketch and $\mathcal{S}_1$ is the compressed sketch. $h(.)$ are hash functions for $\mathcal{S}_0$ and $h'(.)$ are hash functions for $\mathcal{S}_1$.

3.2.3 *Compressing a Sketch of Any Size.*

In section 3.1.1, we assume that the size of the sketch to compress is a power of two. In fact, our algorithm, *i.e.*`hash shifting` and `max selection`, also works when the sketch size is not a power of two. And in this case, our compression algorithms just need to make small modifications.

Given a sketch $\mathcal{S}_0$ to compress, it has d hash functions, $h_1(.)$, $h_2(.)$, ..., $h_d(.)$, and $\mathcal{L}$ counters. Let γ be the compression ratio. To compress it, we first divide the $\mathcal{L}$ counters in $\mathcal{S}_0$ into $\lceil \frac{\mathcal{L}}{\gamma} \rceil$ groups. For the i-th ($0 \leqslant i < \lceil \frac{\mathcal{L}}{\gamma} \rceil - 1$) group, it contains γ counters: $\mathcal{S}_0[i\gamma]$, $\mathcal{S}_0[i\gamma + 1]$, ..., $\mathcal{S}_0[(i + 1)\gamma - 1]$. For the last group, it may contain less than γ counters: $\mathcal{S}_0[\lceil \frac{\mathcal{L}}{\gamma} - 1\rceil\gamma]$, $\mathcal{S}_0[\lceil \frac{\mathcal{L}}{\gamma} - 1\rceil\gamma + 1]$, ..., $\mathcal{S}_0[\mathcal{L} - 1]$. Then, for each group, we choose the sum or the maximum value. When querying an item e, we first calculate d values on e,$h_1(e)\%\mathcal{L}$, $h_2(e)\%\mathcal{L}$, ..., $h_d(e)\%\mathcal{L}$, where % is the modulo operation. Then, using the `hash shifting` technique, we shift the d values right by α bits where $\gamma = 2^\alpha$, and locate d counters in $\mathcal{S}_1$ to estimate the frequency of e.

Example: As shown in Figure 6, the original sketch $\mathcal{S}_0$ has 15 counters, and we use 4-bit hash values to locate counters in $\mathcal{S}_0$. The three hash values on e are $h_1(e) = (0100)_2$, $h_2(e) = (0111)_2$, $h_3(e) = (1110)_2$, and e is hashed to three counters whose values are 7, 12, 10, respectively. When the compression ratio is 4, we divide the counters into four groups, where each of the first three groups has 4 counters and the last group has 3 counters. We use the values of the largest counters in each group to form new counters and finally get $\mathcal{S}_1$ with four new counters. When querying the frequency of e, we shift the three 4-bit hash values right by 2 bits, and locate two (three indeed, $h'_1(e) = h'_2(e)$) counters whose values are 12 and 10, respectively. Thus, the estimated frequency given by $\mathcal{S}_1$ is $min\{12, 10\} = 10$.

4 MATHEMATICAL ANALYSIS

Our ACC algorithm is generic and can be applied to many classical sketch algorithms. In the following part, we apply ACC algorithm to sketches of CM, CU and Count, and prove that: 1) after compression using our ACC algorithm, the characteristic of one-side error of sketches can be kept, which can be seen in section 7.3 in appendix. 2) after compression using our ACC algorithm, the compressed sketches are more accurate than standard sketches using the same bandwidth.

4.1 Proofs of Compressed CM Sketch

We first give the error bound of a standard CM sketch. Consider a Count-min sketch with d arrays and w counters per array. According to the literature [2], we can easily get that the error bound of this CM sketch is:

$$Pr\{\hat{n}_j \geqslant n_j + \varepsilon N\} \leqslant \left(\frac{1}{\varepsilon w}\right)^d \tag{2}$$

where ε is a given positive number, N is the number of distinct recorded items, n_j is the real frequency of item e_j and $\hat{n}_j$ is the estimated frequency of e_j. The left side of the equation means the probability that the error in answering a query is within εN.

Next, we give the error bound of a compressed CM sketch using ACC$_2$.

Theorem 4.1. *Consider a Count-min sketch with d arrays and $w \cdot 2^k$ counters per array. The CM sketch is compressed into $1/2^k$ of the original size. Given an arbitrary small positive number ε and an item e_i, the error bound of the compressed CM sketch is:*

$$\begin{aligned} &Pr\{\hat{n}_j \geqslant n_j + \varepsilon N\} \\ \leqslant &\left\{1 - \left(1 - \frac{1}{\varepsilon w 2^k}\right)\left[1 - \frac{N}{(n_j + \varepsilon N) w 2^k}\right]^{2^k - 1}\right\}^d \end{aligned} \tag{3}$$

where N is the number of distinct recorded items, n_j is the real frequency of item e_j and $\hat{n}_j$ is the estimated frequency of e_j.

Proof. CM sketch has several counter arrays, and the compression method ACC$_2$ is applied to each of these counter arrays. Because they are independent, we just focus on the first counter array. For an item e_j, its true frequency is n_j, and it is mapped to one counter in the first array. Assuming the value of this counter is $V_1 + n_j$, where V_1 is the number of items mapped to this counter excluding e_j. And this counter is in a compression group with other $2^k - 1$ counters, and the value of these $2^k - 1$ counters are V_2, V_3 $\cdots$, V_{2^k}. When compressing, we choose the largest one of these 2^k counters. After compression, the estimated frequency of e_j in the first array, $\hat{n}^1_j$, is:

$$\hat{n}^1_j = max(V_1 + n_j, V_2, V_3, \cdots, V_{2^k}) \tag{4}$$

Therefore, we have:

$$\begin{aligned} &Pr\{\hat{n}^1_j \geqslant n_j + \varepsilon N\} \\ &= 1 - Pr\{max(V_1 + n_j, V_2, V_3, \cdots, V_{2^k}) < n_j + \varepsilon N\} \\ &= 1 - Pr\{V_1 < \varepsilon N\} \prod_{i=2}^{2^k} Pr\{V_i < n_j + \varepsilon N\} \end{aligned} \tag{5}$$

According to Markov inequality, we can get that:

$$Pr\{V_1 < \varepsilon N\} \geqslant 1 - \frac{E(V_1)}{\varepsilon N} \approx 1 - \frac{1}{\varepsilon w 2^k} \tag{6}$$

$$\begin{aligned} Pr\{V_i < n_j + \varepsilon N\} &\geqslant 1 - \frac{E(V_i)}{n_j + \varepsilon N} \\ &\approx 1 - \frac{N}{(n_j + \varepsilon N) w 2^k} \end{aligned} \tag{7}$$

where $i = 2, 3, \cdots, 2^k$. Note that for $1 \leqslant i \leqslant 2^k$, $E(V_i) = \frac{N - n_j}{w 2^k} \approx \frac{N}{w 2^k}$. Therefore, we have:

$$\begin{aligned} &Pr\{\hat{n}^1_j \geqslant n_j + \varepsilon N\} \\ &\leqslant 1 - \left(1 - \frac{1}{\varepsilon w 2^k}\right)\left[1 - \frac{N}{(n_j + \varepsilon N) w 2^k}\right]^{2^k - 1} \end{aligned} \tag{8}$$

Because we choose the minimum one of $\hat{n}^i_j$, where $i = 1, 2, \cdots, d$, as the final estimated frequency for item e_j, *i.e.*$\hat{n}_j = min(\hat{n}^1_j, \hat{n}^2_j, \cdots, \hat{n}^d_j)$, the error bound should be:

$$\begin{aligned} Pr\{\hat{n}_j \geqslant n_j + \varepsilon N\} &= Pr\{min(\hat{n}^1_j, \hat{n}^2_j, \cdots, \hat{n}^d_j) \geqslant n_j + \varepsilon N\} \\ \leqslant &\left\{1 - \left(1 - \frac{1}{\varepsilon w 2^k}\right)\left[1 - \frac{N}{(n_j + \varepsilon N) w 2^k}\right]^{2^k - 1}\right\}^d \end{aligned} \tag{9}$$

□

After getting the two error bounds, we compare these two error bounds and learn that:

Theorem 4.2. *Given a compressed CM sketch using ACC$_2$, where the compression ratio is 2^k. It has the same d (number of counter arrays) and w (number of counters per array) with a standard CM sketch. The compressed CM sketch has a lower error bound than the standard CM sketch.*

Proof. We use P_C to denote the error bound of the compressed CM sketch using ACC$_2$, and use P_S to denote the error bound of a standard CM sketch with the same d and w. Then, we have:

$$\begin{aligned} P_C &= \left\{1 - \left(1 - \frac{1}{\varepsilon w 2^k}\right)\left[1 - \frac{N}{(n_j + \varepsilon N) w 2^k}\right]^{2^k - 1}\right\}^d \\ &< \left[1 - \left(1 - \frac{1}{\varepsilon w 2^k}\right)^{2^k}\right]^d \end{aligned} \tag{10}$$

We define a function $F(x) = \left(1 - \frac{w_0}{x}\right)^x$, where $w_0 = \frac{1}{\varepsilon w}$. According to the inequality of arithmetic, we have:

$$F(x) = 1 \cdot \left(1 - \frac{w_0}{x}\right)^x \leqslant \left[\frac{1 + (1 - \frac{w_0}{x})x}{1 + x}\right]^{x+1} = F(x+1) \tag{11}$$

Therefore, $F(x)$ is a monotonic increasing function. So:

$$P_C < \left[1 - F(2^k)\right]^d \leqslant [1 - F(1)]^d = w_0^d = P_S \tag{12}$$

Thus, the error bound of the compressed CM sketch using ACC$_2$ is smaller than the error bound of the standard CM sketch with the same d and w. □

4.2 Proofs of Compressed CU Sketch

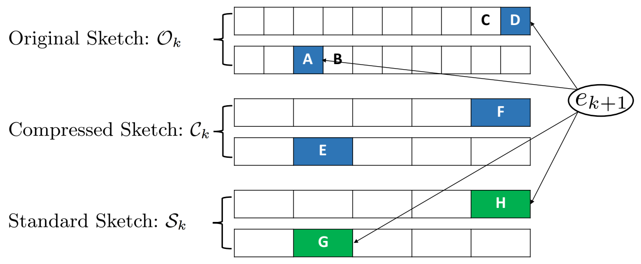
The difference between CM sketch and CU sketch is that CU sketch increases only the smallest counters that are mapped to in the recording processing. Therefore, the value of a counter in CU sketch is not only determined by the number of the items that are mapped to this counter, but also determined by the order in which they come. For this reason, we use mathematical induction to prove that a compressed CU sketch using ACC$_2$ is more accurate than a standard CU sketch with same size.

Theorem 4.3. *Given a compressed CU sketch using ACC$_2$, where the compression ratio is 2. It has the same d (number of counter arrays) and w (number of counters per array) with a standard CU sketch. Let n be the number of items recorded. For any $n \in N$,*

$$C_n[i][j] \leqslant S_n[i][j]$$

where $C_n[i][j]$ is the j-th counter in the i-th array in the compressed sketch after recording n items, and $S_n[i][j]$ is the j-th counter in the i-th array in the standard sketch after recording n items.

Proof. Assuming that the original sketch recording n items before compression is O_n, we have $C_n[i][j] = max(O_n[i][2j], O_n[i][2j+1])$.
Base case $n = 1$: If $n = 1$, each counter array in O_1 has only one counter with value 1 and the rest of the counters are 0. For the i-th counter array, assuming $O_1[i][j] = 1$, so $C_1[i][\lfloor \frac{j}{2} \rfloor] = 1$ and $S_1[i][\lfloor \frac{j}{2} \rfloor] = 1$. Therefore, the theorem holds when $n = 1$.
Inductive hypothesis: Suppose the theorem holds for all values of n up to some k, $k \geqslant 1$, which means that $C_n[i][j] \leqslant S_n[i][j]$ for all $n \leqslant k$.
Inductive step: Let $n = k + 1$. As shown in Fig. 7, the k+1-th item is mapped to $\{A, D\}$ in O_k and $\{G, H\}$ in S_k. Then, the smaller counters in $\{A, D\}$ and $\{G, H\}$ are increased by 1. There are four cases when recording e_{k+1}:



Original Sketch: $\mathcal{O}_k$
C
D
A
B
e_{k+1}
Compressed Sketch: $\mathcal{C}_k$
F
E
Standard Sketch: $\mathcal{S}_k$
H
G

Figure 7: To record the k+1-th item e_{k+1}, it is mapped to two counters in O_k and S_k, respectively. After compression, the larger counters in $\{A, B\}$ and $\{C, D\}$ form two new counters, E and F.

case 1: *A is no larger than D and G is no larger than H.* Then $A' = A + 1$ and $G' = G + 1$. Therefore, we have:

$$E' = max(A', B) \leqslant max(A, B) + 1 = E + 1 \leqslant G + 1 = G'.$$

case 2: *A is no larger than D and G is larger than H.* Then $A' = A + 1$ and $G' = G \geqslant H + 1$. Therefore, we have:

$$E' = max(A', B) = max(A + 1, B) \leqslant max(D + 1, B).$$

Noticing that $B \leqslant E \leqslant G$ and $D \leqslant F \leqslant H$, we have

$$E' \leqslant max(D + 1, B) \leqslant max(H + 1, G) \leqslant max(G, G) = G = G'.$$

case 3: *A is larger than D and G is no larger than H.* Then $A' = A$ and $G' = G + 1$. Therefore, we have:

$$E' = max(A', B) = max(A, B) = E \leqslant G < G'.$$

case 4: *A is larger than D and G is larger than H.* Then $A' = A$ and $G' = G$. Therefore, we have:

$$E' = max(A', B) = max(A, B) = E \leqslant G = G'.$$

So, the theorem holds for $n = k + 1$.

By the principle of mathematical induction, the theorem holds for all $n \in N$. □

According to *theorem 4.1* and *theorem 4.5*, we know that the compressed CU sketch using ACC$_2$ is more accurate than a standard CU sketch with same size.

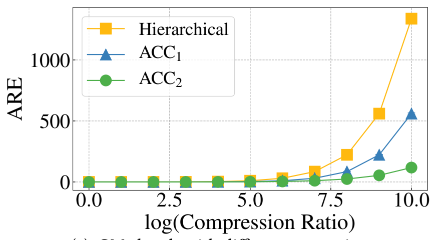
4.3 Proofs of Compressed Count Sketch

Compared to CM sketch and CU sketch, Count sketch randomly increases or decreases the counters, which cannot guarantee the characteristic of one-side error. In this case, ACC$_2$ does not work well. Fortunately, ACC$_1$ still work.

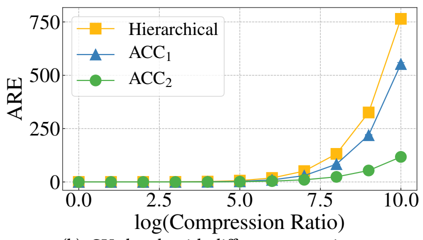
Let $S_0, S_1, S_d, \mathcal{I}', \mathcal{I}'', \gamma, n_k, e_k$ be subject to the definition in 7.3 and 7.4, $h'(.)$ be the hash function that determines whether increments or decrements the counter value, then

$$\begin{aligned} S_1[i] &= \sum_{r=i\gamma}^{(i+1)\gamma - 1} S_0[r] = \sum_{e_k \in \mathcal{I}'} h'(e_k) \cdot n_k \\ S_d[i] &= \sum_{e_k \in \mathcal{I}''} h'(e_k) \cdot n_k \end{aligned} \tag{13}$$

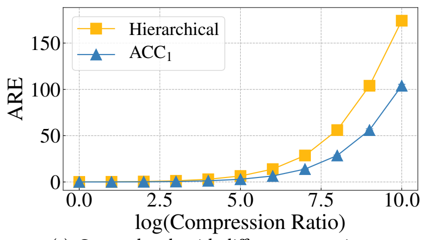
Since we have proved that $\mathcal{I}' = \mathcal{I}''$, all corresponding counters in both sketches satisfy that $S_1[i] = S_d[i]$. So the Count sketch S_1 using ACC$_1$ has the same accuracy as a standard Count sketch S_d with the same size.



(a) CM sketch with different strategies.

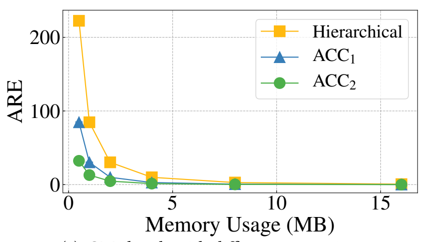


(b) CU sketch with different strategies.

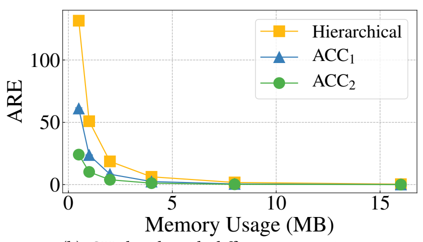


(c) Count sketch with different strategies.

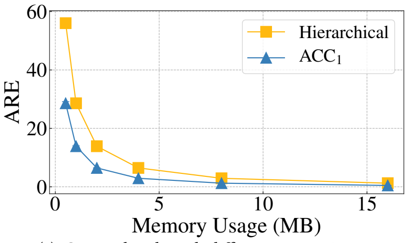
Figure 8: ARE vs. Compression Ratio



(a) CM sketch with different strategies.

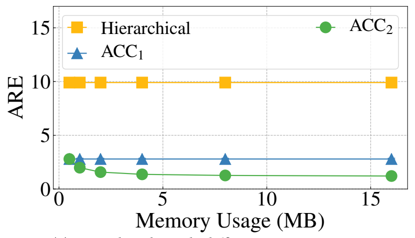


(b) CU sketch with different strategies.

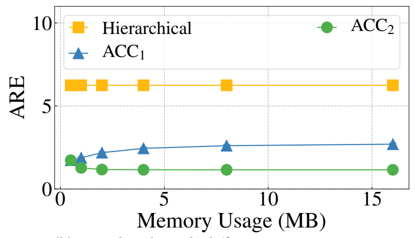


(c) Count sketch with different strategies.

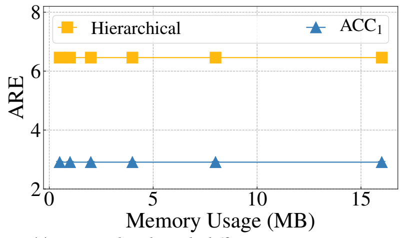
Figure 9: ARE vs. Memory Usage



(a) CM sketch with different strategies.



(b) CU sketch with different strategies.



(c) Count sketch with different strategies.

Figure 10: ARE of sketches with same size after compression.

5 EXPERIMENTAL RESULTS

The detailed experimental setup and evaluation metrics can be seen in section 7.1 and 7.2 in appendix. We first show the accuracy of our algorithm and other algorithms. Then, we compare the processing speed of our algorithm and other algorithms. Next, we show the ability of different algorithms to adapt to bandwidth. Finally, we show the experimental results on different datasets.

5.1 Experiments on Accuracy

5.1.1 Accuracy vs. Compression Ratio. In this experiment, we allocate the same memory, 16MB, and change the compression ratio.

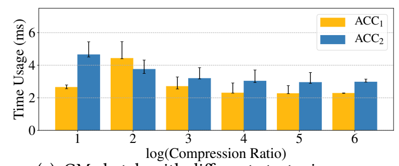
ARE vs. Compression ratio (Figure 8): According to Figure 8(a), the results show that when the compression ratio is 2^{10}, the ARE of CM sketch with ACC_2 is about 11.34 and 4.78 times lower than hierarchical CM sketch and CM sketch with ACC_1, respectively. According to Figure 8(b), the results show that when the compression ratio is 2^{10}, the ARE of CU sketch with ACC_2 is about 6.52 and 4.74 times lower than hierarchical CU sketch and CU sketch with ACC_1, respectively. According to Figure 8(c), the results show that when the compression ratio is 2^{10}, the ARE of Count sketch with ACC_1 is about 1.71 times lower than hierarchical Count sketch.

5.1.2 Accuracy vs. Memory Usage. In this experiment, we allocate different memory (0.5, 1.0, 2.0, 4.0, 8.0, 16.0MB) and set the compression ratio to 8.

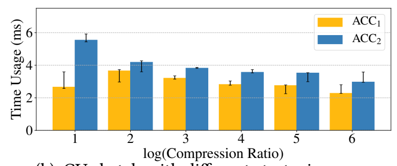
ARE vs. Memory Usage (Figure 9): According to Figure 9(a), the results show that when the memory usage is 2.0 MB, the ARE of CM sketch with ACC_2 is about 6.42 and 2.13 times lower than hierarchical CM sketch and CM sketch with ACC_1, respectively. According to Figure 9(b), the results show that when the memory usage is 2.0 MB, the ARE of CU sketch with ACC_2 is about 4.77 and 2.16 times lower than hierarchical CU sketch and CU sketch with ACC_1, respectively. According to Figure 9(c), the results show that when the memory usage is 2.0 MB, the ARE of Count sketch with ACC_1 is about 2.15 times lower than hierarchical Count sketch.

5.1.3 Accuracy vs. Memory Usage and Compression Ratio. In this experiment, we allocate different memory (0.5MB, 1.0MB, 2.0MB, 4.0MB, 8.0MB, 16.0MB) to sketches with different strategies. And we keep these sketches having same size (0.5MB) after compression by adjusting the compression ratio.

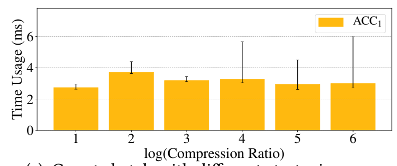
ARE (Figure 10): According to Figure 10(a), the results show that when the memory usage is 2.0 MB, the ARE of CM sketch with ACC_2 is about 6.30 and 1.78 times lower than hierarchical CM sketch and CM sketch with ACC_1, respectively. According to Figure 10(b), the results show that when the memory usage is 2.0 MB, the ARE of CU sketch with ACC_2 is about 5.31 and 1.87 times lower than hierarchical CU sketch and CU sketch with ACC_1, respectively. According to Figure 10(c), the results show that when the memory



(a) CM sketch with different strategies.

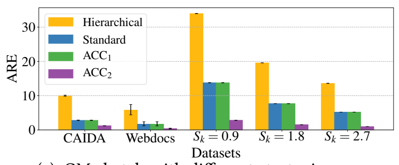


(b) CU sketch with different strategies.

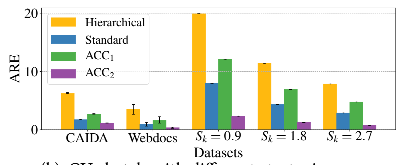


(c) Count sketch with different strategies.

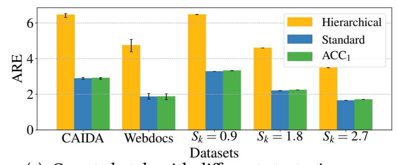
Figure 11: The compression speed of sketches using ACC$_1$ and ACC$_2$.



(a) CM sketch with different strategies.

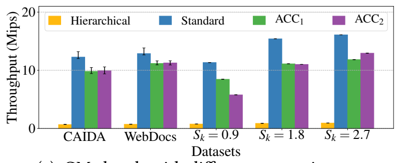


(b) CU sketch with different strategies.

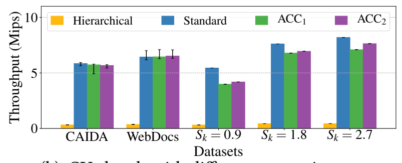


(c) Count sketch with different strategies.

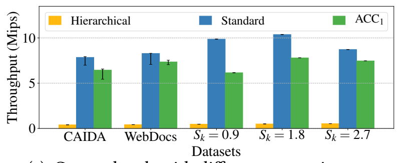
Figure 12: ARE vs. Datasets



(a) CM sketch with different strategies.



(b) CU sketch with different strategies.



(c) Count sketch with different strategies.

Figure 13: Insertion Speed vs. Datasets

usage is 2.0 MB, the ARE of Count sketch with ACC$_1$ is about 2.10 times lower than hierarchical Count sketch.

5.2 Experiments on Compression Speed

Compression Speed: In this experiment, we implement ACC$_1$ algorithm on sketches of CM, CU and Count, and implement ACC$_2$ algorithm on CM and CU sketch. We change the compression ratio and test the compression speed. According to Figure 11(a) and 11(b), the results show that the time usage for compressing a CM or a CU sketch using ACC$_1$ is about 3 $\sim$ 5 ms, and the time usage for compressing a CM or a CU sketch using ACC$_2$ is about 3 $\sim$ 6 ms. According to Figure 11(c), the results show that the time usage for compressing a Count sketch using ACC$_1$ is about 2 $\sim$ 4 ms. Though the compression speed of ACC$_2$ is a bit slower than ACC$_1$, ACC$_2$ can compress a 16MB sketch in 6 ms. Therefore, we think that ACC$_2$ is fast enough.

Insertion Speed: In this experiment, we implement hierarchical strategy and ACC$_1$ on sketches of CM, CU and Count, and implement ACC$_2$ on sketches of CM and CU. According to Figure 14, the insert throughput of hierarchical CM sketch is 14.9 times lower than the throughput of the CM sketch using ACC$_1$ and ACC$_2$. The insert throughput of hierarchical CU sketch is 18.3 times lower than the throughput of the CU sketch using ACC$_1$ and ACC$_2$. The insert throughput of hierarchical Count sketch is 16.1 times lower than the throughput of the Count sketch using ACC$_1$.

5.3 Experiments on Adaptability to Bandwidth

In this experiment, we implement standard CM, CU, Count sketch, and implement hierarchical strategy and ACC$_1$ on CM, CU and Count sketch, and implement ACC$_2$ on CM and CU sketch. For standard CM, CU and Count sketch, we change their memory usage so that the AREs of the reporting results of them are lower than

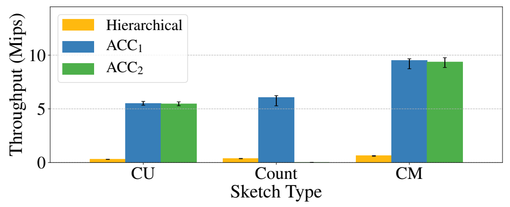


Figure 14: Insertion speed of sketches.

0.26. Besides these standard sketches, we set the memory usage of other data structures to 16 MB, and change the compression ratio so that the AREs of the reporting results of these data structures are lower than 0.26. According to Figure 15, the experimental results show that, in order to make the ARE lower than 0.26, the bandwidth need of CM sketch using ACC$_2$ is 1.39, 2 and 2 times lower than standard CM sketch, hierarchical CM sketch and CM sketch using ACC$_2$, respectively. The bandwidth need of CU sketch using ACC$_2$ is 1.11, 2 and 2 times lower than standard CU sketch, hierarchical CU sketch and CU sketch using ACC$_1$, respectively. The bandwidth need of Count sketch using ACC$_1$ is exactly the same as hierarchical Count sketch. Due to the bandwidth for sketches using ACC$_1$ is discrete, the bandwidth for Count sketch using ACC$_1$ is a bit higher than standard Count sketch.

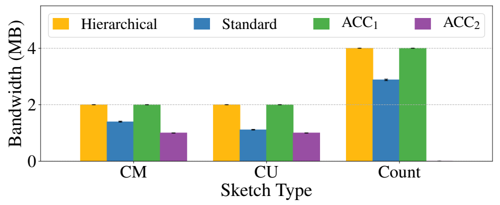


Figure 15: The bandwidth needed for sketches using different strategies to achieve that the AAE of the sketches are below 0.26

5.4 Experiments on Different Datasets

In this experiment, we implement hierarchical strategy and ACC_1 on CM, CU and Count sketch, and implement our ACC_2 on CM and CU sketch. We conduct experiments on three kinds of datasets and show the AAE, ARE, C_r, and insertion speed.

According to Figure 12(a) and 12(b), the experimental results show that, on the three kind of datasets, the AREs of CM/CU sketches using ACC_2 are lower than AREs of standard CM/CU sketch, hierarchical CM/CU sketches and CM/CU sketches using ACC_1. According to Figure 12(c), the ARE of Count sketch using ACC_1 is lower than the ARE of hierarchical Count sketch, and is the same as the standard Count sketch.

According to Figure 13(a) and 13(b), the insert throughput of the CM/CU sketch using ACC_1 and ACC_2 is much higher than the insert throughput of hierarchical CM/CU sketch, and is lower than the throughput of standard CM/CU sketch. According to Figure 13(c), the insert throughput of count sketch using ACC_1 is much higher than the insert throughput of hierarchical count sketch, and is lower than the throughput of standard count sketch.

6 CONCLUSION

Hash based data structure such as sketch, bloom filter and hash table, are important work in data stream area. In this paper, we propose ACC_1 and ACC_2 algorithms for compressing sketches so that they can be adapted to available bandwidth while keeping high accuracy. We applied our compression algorithm to three typical sketches: CM, CU and Count sketch. Experimental results show that our ACC_1 and ACC_2 algorithms are fast and accurate, and can help sketches adapt to available bandwidth. We believe that our ACC algorithm can be applied to more sketches and more areas. All related source codes are released on Github [1].

REFERENCES

[1] Related source codes. https://github.com/CompressedSketch/CompressedSketch.

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

[3] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[4] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, pages 784–784, 2002.

[5] Tao Li, Shigang Chen, and Yibei Ling. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM Transactions on Networking (TON)*, 20(5):1622–1634, 2012.

[6] Tong Yang, Lingtong Liu, Yibo Yan, Muhammad Shahzad, Yulong Shen, Xiaoming Li, Bin Cui, and Gaogang Xie. Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 103–106. IEEE, 2017.

[7] Sumit Ganguly, Phillip B Gibbons, Yossi Matias, and Avi Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *ACM SIGMOD Record*, volume 25, pages 271–281. ACM, 1996.

[8] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. Pyramid sketch: A sketch framework for frequency estimation of data streams. *Proceedings of the VLDB Endowment*, 10(11):1442–1453, 2017.

[9] Haiqin Liu, Yan Sun, and Min Sik Kim. Fine-grained ddos detection scheme based on bidirectional count sketch. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–6. IEEE, 2011.

[10] Yeonhee Lee and Youngseok Lee. Detecting ddos attacks with hadoop. In *Proceedings of The ACM CoNEXT Student Workshop*, page 7. ACM, 2011.

[11] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1093–1103. Association for Computational Linguistics, 2012.

[12] Amit Goyal and Hal Daumé III. Approximate scalable bounded space sketch for large data nlp. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 250–261. Association for Computational Linguistics, 2011.

[13] Amit Goyal, Jagadeesh Jagarlamudi, Hal Daumé III, and Suresh Venkatasubramanian. Sketching techniques for large scale nlp. In *Proceedings of the NAACL HLT 2010 Sixth Web as Corpus Workshop*, pages 17–25. Association for Computational Linguistics, 2010.

[14] Veronica Mayorga and Neoklis Polyzotis. Sketch-based summarization of ordered xml streams. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 541–552. IEEE, 2009.

[15] Florin Rusu and Alin Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 187–198. ACM, 2007.

[16] Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 234–243. ACM, 2003.

[17] Cheqing Jin, Weining Qian, Chaofeng Sha, Jeffrey X Yu, and Aoying Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 287–294. ACM, 2003.

[18] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Processing data-stream join aggregates using skimmed sketches. In *International Conference on Extending Database Technology*, pages 569–586. Springer, 2004.

[19] Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Practical algorithms for tracking database join sizes. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 297–309. Springer, 2005.

[20] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and Yuankun Zhong. Finding persistent items in data streams. *Proceedings of the VLDB Endowment*, 10(4):289–300, 2016.

[21] Nitin Agrawal and Ashish Vulimiri. Low-latency analytics on colossal data streams with summarystore. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 647–664. ACM, 2017.

[22] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 795–810. ACM, 2015.

[23] Yanqing Peng, Jinwei Guo, Feifei Li, Weining Qian, and Aoying Zhou. Persistent bloom filter: Membership testing for the entire history. 2018.

[24] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proceedings of the VLDB Endowment*, 6(10):937–948, 2013.

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

[26] Qi George Zhao, Mitsunori Ogihara, Haixun Wang, and Jun Jim Xu. Finding global icebergs over distributed data sets. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 298–307. ACM, 2006.

[27] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proceedings of the VLDB Endowment*, 5(10):992–1003, 2012.

[28] Donald E Knuth. Dynamic huffman coding. *Journal of algorithms*, 6(2):163–180, 1985.

[29] Mark R Nelson. Lzw data compression. *Dr. Dobb's Journal*, 14(10):29–36, 1989.

[30] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 123–137. ACM, 2015.

[31] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208. ACM, 2009.

[32] Jiecao Chen and Qin Zhang. Bias-aware sketches. *Proceedings of the VLDB Endowment*, 10(9):961–972, 2017.

[33] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proc. SIGMOD*, 2016.

[34] The CAIDA Anonymized Internet Traces 2016. http://www.caida.org/data/overview/.

[35] Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data/.

[36] Alex Rousskov and Duane Wessels. High-performance benchmarking with web polygraph. *Software: Practice and Experience*, 2004.

[37] David MW Powers. Applications and explanations of zipf's law. In *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*, pages 151–160. Association for Computational Linguistics, 1998.

[38] Bob hash website. http://burtleburtle.net/bob/hash/evahash.html.

7 APPENDIX

7.1 Experimental Setup

Three kinds of datasets are used for evaluation as shown in Table 1.
CAIDA: The first kind of datasets is anonymized IP traffic streams collected in Equinix-Chicago monitor from CAIDA [34]. The items in the traffic stream are flows identified by their source IP addresses. The traffic trace contains 10-minute traffic data, and is split into 5-second traces.
Web Page: The second kind of datasets is obtained from the website *Frequent Itemset Mining Dataset Repository* [35], which is crawled from a collection of web pages. The size of the items in the datasets is 4 bytes, recording the number of distinct terms in a web page. We form eight 64MB sub-datasets from the raw datasets, each of which has about 0.9M distinct items.
Synthetic: The third kind is synthetic datasets generated by the code from Web Polygraph [36], which is a performance testing tool. These synthetic datasets follow the Zipf [37] distribution and possess a crucial coefficient named *skewness* ranging from 0 to 3.0 with a step of 0.3.

Dataset	# trace	# total items	#distinct items
CAIDA	11	2.5M	107.2K~109.6K
WebPage	9	2.5M	87.4K~130.7K
Synthetic	11	2.5M	207.4K~917.7K

Table 1: Datasets for evaluation.

We implement the CM [2], CU [3], Count sketch [4] with C++, and use *hierarchical strategy*, *ACC*$_1$, *ACC*$_2$ method on them. Since the Count sketch can not guarantee one-side error, we didn't apply ACC$_2$ to the Count sketch in the experiments. When using hierarchical sketches, for recording, we set the number of counters in each counter array to a power of 2. For sending, we compress the hierarchical sketch into a smaller sketch with hierarchical structure. In terms of hash functions, we use Bob hash functions recommended by [38]. We performed all these experiments on a machine with 12-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62 GB total DRAM memory.

7.2 Evaluation Metrics

To evaluate the accuracy of sketches, there are three typical metrics: average absolute error, average relative error, and correct rate.

- **AAE (Average Absolute Error)**: AAE is defined as $\frac{1}{n}\sum_{i=1}^{n}|f_i - \hat{f_i}|$, where n is the number of different items, and f_i and $\hat{f_i}$ are the real and estimated frequencies of item e_i, respectively.
- **ARE (Average Relative Error)**: ARE is defined as $\frac{1}{n}\sum_{i=1}^{n}\left(\frac{|f_i-\hat{f_i}|}{f_i}\right)$, where n is the number of different items, and f_i and $\hat{f_i}$ are the real and estimated frequencies of item e_i, respectively.
- **Throughput**: We simulate how sketches actually insert and query on CPU platform, and calculate the throughput for each insertion or compression. Its unit here is million instructions per second (Mips). To minimize the accidental derivation, we repeat each experiments 100 times.

7.3 One-side Error Guarantee

In this section, we prove that our compression algorithms ACC$_1$ and ACC$_2$ can keep the characteristic of one-side error of sketches.

THEOREM 7.1. *Given any sketch with one-side error, after compression by ACC*$_1$ *or ACC*$_2$*, it can be guaranteed that the compressed sketch also has one-side error.*

PROOF. Given γ adjacent counters $S_0[i\gamma], \cdots, S_0[(i+1)\gamma-1]$ in the sketch S_0. Let $\mathcal{I}_{i\gamma}, \cdots, \mathcal{I}_{(i+1)\gamma-1}$ be the sets of elements that are mapped to these counters, respectively. If S_0 has one-side error, for any $i\gamma \leqslant j \leqslant (i+1)\gamma - 1$ and $e_k \in \mathcal{I}_j$, $S_0[j] \geqslant n_k$ holds, where n_k is the real frequency of item e_k.

Let S_1 be the compressed sketch using ACC$_1$, and $\mathcal{I}' = \bigcup_{r=i\gamma}^{(i+1)\gamma-1}\mathcal{I}_r$ be the set of elements mapped to the counter $S_1[i]$. Therefore, for any $e_k \in \mathcal{I}'$, there is a set $\mathcal{I}_j$ such that $e_k \in \mathcal{I}_j$, and

$$S_1[i] = \sum_{r=i\gamma}^{(i+1)\gamma-1} S_0[r] \geqslant S_0[j] \geqslant n_k \tag{14}$$

Let S_2 be the compressed sketch using ACC$_2$. Similarly, we have

$$S_2[i] = \max_{r=i\gamma}^{(i+1)\gamma-1} S_0[r] \geqslant S_0[j] \geqslant n_k \tag{15}$$

From equation (14) and (15), we know that for any element mapped to a new counter, the estimated frequency must be greater than its true frequency. In other words, the compressed sketches keep the characteristic of one-side error. □

7.4 Accuracy Consistency using ACC$_1$

Given a sketch S_0 of size 2^s, let S_1 be the compressed sketch using *ACC*$_1$ with the compression ratio $\gamma = 2^{\alpha}$, and S_d be the standard sketch of size $2^{s-\alpha}$. We can prove that S_1 and S_d are the same in some cases.

Without loss of generality, assume that the number of hash functions of the sketch is 1. Let $h_0(.)$ be the hash function used by S_0, $h_2(.)$ be the hash function used by S_d.

THEOREM 7.2. *If* $h_2(x) = h_0(x) \gg \alpha$ *holds,* S_1 *and* S_d *are totally the same.*

PROOF. Let $\mathcal{I}', e_k, n_k$ be subject to the definition in 7.3, and $\mathcal{I}''$ be the set of elements that are mapped to $S_d[i]$, then

$$\begin{aligned} S_1[i] &= \sum_{r=i\gamma}^{(i+1)\gamma-1} S_0[r] = \sum_{e_k\in\mathcal{I}'} n_k \\ S_d[i] &= \sum_{e_k\in\mathcal{I}''} n_k \end{aligned} \tag{16}$$

Notice that the value of the hash function,

$$\begin{aligned} \mathcal{I}' &= \{e_k \mid h_0(e_k) \in [i\gamma, (i+1)\gamma)\} \\ \mathcal{I}'' &= \{e_k \mid h_d(e_k) = i\} = \{e_k \mid (h_0(e_k) \gg \alpha) = i\} \end{aligned} \tag{17}$$

Since $\gamma = 2^{\alpha}$, $x \gg \alpha = \lfloor \frac{x}{\gamma} \rfloor$. Obviously, we have $x \in [i\gamma, (i+1)\gamma)$, so we can get $\lfloor \frac{x}{\gamma} \rfloor = i$. Thus, $\mathcal{I}' = \mathcal{I}''$, and $S_1[i] = S_d[i]$. Due to the arbitrariness of i, S_1 and S_d are totally the same. □