Fast and Accurate Cardinality Estimation by Self-Morphing Bitmaps

Haibo Wang, *Graduate Student Member, IEEE*, Chaoyi Ma, *Graduate Student Member, IEEE*, Shigang Chen, *Fellow, IEEE*, and Yuanda Wang

***Abstract*— Estimating the cardinality of a data stream is a fundamental problem underlying numerous applications such as traffic monitoring in a network or a datacenter and query optimization of Internet-scale P2P data networks. Existing solutions suffer from high processing/query overhead or memory in-efficiency, which prevents them from operating online for data streams with very high arrival rates. This paper takes a new solution path different from the prior art and proposes a self-morphing bitmap, which combines operational simplicity with structural dynamics, allowing the bitmap to be morphed in a series of steps with an evolving sampling probability that automatically adapts to different stream sizes. We further generalize the design of self-morphing bitmap. We evaluate the self-morphing bitmap theoretically and experimentally. The results demonstrate that it significantly outperforms the prior art.**

***Index Terms*— Cardinality estimation, bitmap, morphing.**

I. Introduction

CARDINALITY estimation is one of the fundamental problems in the data steaming field [1]–[10]. A data stream is a sequence of data items arriving at high rate. Its cardinality is the number of *distinct* data items in the stream. The term data item under different practical scenarios may refer to an attribute (such as source address) of each packet in an Internet traffic stream [11]–[15], node ID in wireless sensor networks for in-network query aggregation [16], each file (name) accessed by users in P2P networks [17], etc.

Cardinality estimation has many practical applications. Consider Internet traffic received by a router and the application of anomaly detection [18], [19]. We may treat all packets sent from the same source address as a data stream. Each packet carries a data item, which may be the destination address in the packet header. The cardinality of a stream is the number of destination addresses that the source address has contacted. By measuring the cardinality for the packet stream from each source, a cardinality estimation module deployed at the gateway of an enterprise network can detect external scanners, i.e., those that contact too many distinct destination addresses. In another example, we may define all packets sent to the same destination address as a data stream, where the source address in the packet header as data item. If we observe that the cardinality of a stream surges, it may signal a DDoS attack as an abnormal number of distinct sources access the service hosted at the destination address.

The problem of cardinality estimation is challenging because each item may appear in the stream multiple times and we must filter out duplicate items, which will require us to "remember" the items that have been counted. As a stream may contain millions or even billions of different items, it can be costly to remember all items and find out duplicates. This paper studies efficient data structures and algorithms that can estimate the cardinality of a stream with high accuracy, low memory overhead, and low processing overhead. Consider a modern router with a line rate of hundreds of gigabits or even terabits per second. Use the previous example where all packets from each source address form a stream. The number of data streams (source addresses) observed by the router can be in millions. While tracking distinct items in one stream is already a challenge, simultaneously doing that for millions of streams requires a significant amount of SRAM memory on the data plane of the router that processes packets at line rate. Therefore, we need to design the cardinality estimation module both memory efficient and processing efficient in order to record data items at high rate (also referred to as recording throughput). As another justification, various data analysis systems at Google [20], such as Sawzall [21], Dremel [22], and PowerDrill [23], estimate the cardinalities of very large data sets on a daily basis. As pointed out in the paper [20], cardinality estimation over large datasets presents a challenge in terms of computational resources, and memory in particular; for the PowerDrill system, a non-negligible fraction of queries historically could not be computed because they exceeded the available memory.

Real-time applications need to perform cardinality queries online as the data items are recorded. Ideally, in the example of scan detection, for each arrival packet, as we record its destination address for the stream of its source address, we also query for whether the cardinality of the stream exceeds a threshold. However, such per-packet query may not be feasible if query overhead is much larger than recording overhead. In this case, we can only perform queries for some packets

Manuscript received 8 November 2021; accepted 23 January 2022; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Caesar. Date of publication 10 February 2022; date of current version 18 August 2022. This work was supported by NSF under Grant CNS-1719222 and Grant CSR-1909077. *(Corresponding author: Shigang Chen.)*

The authors are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: wanghaibo@ufl.edu; ch.ma@ufl.edu; sgchen@cise.ufl.edu; yuandawang@ufl.edu).

Digital Object Identifier 10.1109/TNET.2022.3147204

at a rate lower than the recording throughput. It is also our goal to improve the rate at which queries can be made (also referred to as query throughput).

Finding the exact cardinality of a data stream is extremely memory/computation-hungry, especially for the current big data streams. It requires either a synopsis size as large as the data stream itself (e.g., sorting) or even more memory (e.g., using hash table). Practically, users may relax the need for an exact solution and may be interested in estimating the cardinality approximately at a much smaller cost, which is what this paper tries to study. There are three categories of solutions to estimating the cardinality of a data stream. The first category performs uniform hash on each data item, keeps the k minimum hash values and produces cardinality estimate based on the k-th minimum value [24], [25]. However, these solutions suffer from high estimation variance and sometimes produce estimates that deviate far from the real cardinality. The second category includes FM [1], LogLog [26], HyperLogLog (HLL) [27] and HyperLogLog++ (HLL++) [20], which use a large number of registers, each providing a coarse cardinality estimation. They take the average of the numerous coarse estimations, trading memory overhead for high accuracy; among them, HLL++ and its variants are most accurate [28]. Their problem is high query overhead and low query throughput, making them unsuitable for online use. The third category is bitmap [2] and its variants, which achieve high query throughput due to their relatively low query overhead. Using bitmap to record data items is not memory-efficient, resulting in small estimation range [2]. One approach is to use sampling to reduce the number of items needed to be recorded. However, the optimal sampling probability is a function of the true cardinality of the stream, which we do not know. The best solution in the literature, i.e., Multi-resolution Bitmap (MRB) [3], [4], maintains multiple bitmaps, each with a different sampling probability. When being queried, MRB finds out which sampling probability is the best based on the current content of all the bitmaps. It then uses only the information stored under that sampling probability to perform cardinality estimation. It wastes information recorded under other sampling probabilities and the memory that is used to record them.

Our goal is to design a new solution for cardinality estimation that significantly outperforms the existing work. For fair comparison, consider all solutions are assigned the same amount of memory. Compared to HLL++ [20] which is most accurate but has low query throughput, the new solution should support much higher query (and recording) throughput, making it suitable for online operations, yet without scarifying accuracy, or even improving accuracy over HLL++. Compared to MRB [3], [4] which is efficient in query throughput but less accurate, the new solution should achieve much better accuracy, yet without scarifying query throughput, or even improving query (and recording) throughput over MRB.

The basic idea in our solution is to use a single self-morphing bitmap (SMB) with a sampling probability that changes over time as more and more data items are recorded. We should use large sampling probabilities for small data streams to ensure accuracy and small sampling probabilities for large data streams to ensure memory efficiency. SMB begins with a sampling probability of 100% and progressively decreases it based on the number of items recorded. In the meanwhile, it morphs the bitmap through a series of steps in such a way that allows us to utilize all recorded information (under different sampling probabilities) for cardinality estimation. SMB uses a single bitmap and a single sampling probability at any given time, whereas MRB uses multiple bitmaps and multiple sampling probabilities. SMB uses all recorded information, whereas MRB uses only some for estimation.

We formally derive the estimation error bound of SMB and evaluate its performance in comparison with the state of the art. The experimental results demonstrate that SMB outperforms in estimation accuracy, recording throughput, and query throughput than the existing work, with significant improvements: (1) It achieves 50% estimation error reduction, comparing with MRB; moreover, its recording/query throughput is higher than MRB. (2) It achieves at least an order of magnitude higher throughput than HLL++; moreover, its accuracy is better than HLL++.

II. Background and Prior Art

A. Problem Statement

A data stream D is a sequence of data items where any data item $d \in D$ may appear once or multiple times. The *stream cardinality* is defined as the number of distinct data items in the stream. For instance, consider $D = \{d_1, d_2, d_1, d_1\}$. Its cardinality is 2 because there are two distinct data items, i.e., d_1 and d_2. The core issue of estimating stream cardinality is to remember the recorded data items, such that duplicate data items will not be counted multiple times. The problem is to design a data structure called *cardinality estimator* that records the data items of a stream and estimates the stream cardinality.

Consider the application of detecting DDoS attacks to a certain internal server, where all packets to the server form a data stream and each data item is distinguished by the source address carried by the packet. If there are 10000 packets from the same source address, the stream cardinality is 1. If the 10000 packets each are sent from different sources addresses, the stream cardinality is 10000. Estimating the stream cardinality can help network reconnaissance against potential DDoS attacks and enable system admin to take further measures.

B. Prior Cardinality Estimators

Cardinality estimation is a classical problem and there are many existing solutions. This subsection first describes the prior cardinality estimators and then compares existing them in terms of recording and query overheads and estimation accuracy. We consider two major computation-hungry operations, i,e., hash and memory access, to assess the recording and query overhead. Constant $\mathcal{H}$ represents the overhead for one hash and $\mathcal{A}$ represents the average overhead for accessing one bit.

Bitmap. Bitmap [2] is an array B of m bits. Upon the arrival of an item d, bitmap performs uniform hash operation $H(\cdot) \in [0, m)$ on d and sets $B[H(d)]$ to 1. The recording

overhead per item is thus $\mathcal{H}+\mathcal{A}$. When being queried, bitmap traverses all the bits and calculates the number of ones in the array, denoted as U, resulting in the query overhead of $m\mathcal{A}$. The cardinality estimate $\hat{n}$ by bitmap [2] is

$$\hat{n} = -m\ln(1-U/m). \tag{1}$$

The maximum useful value of U is $m-1$, which promises the maximum estimate of $m\ln m$. Bitmap provides the best accuracy among all existing cardinality estimators under the condition that there is sufficient memory space, which roughly speaking can be linear to the cardinality and is far too large to be kept in available memory in most applications [29].

Multi-resolution Bitmap (MRB). MRB [3], [4] employs $k > 1$ bitmaps, denoted as $B_0, B_1, \ldots, B_{k-1}$, each with a distinct sampling probability $p_i, \forall 0 \le i < k$, to simultaneously record a data stream and selects one of them for producing the cardinality estimate. Given the total memory allocation of m bits, each ith bitmap B_i is $\frac{m}{k}$ (suppose it is integer) bits long. Considering the convenience of implementation and estimation accuracy, MRB recommends that $p_i = \frac{1}{2^i}, \forall 0 \le i < k$ [3], [4] and $p_0 > p_1 > \ldots > p_{k-1}$. For any data item d, it gets sampled by B_i if $p_i > \frac{H'(d) \pmod W}{W}$, where W is a sufficiently large integer constant and $H'(\cdot)$ is a uniform hash function. Note that if the data item gets sampled by B_i, it gets sampled by $B_0, \ldots, B_{i-1}$ as well. For any data item that only gets sampled by $B_0, B_1, \ldots, B_i$, *instead of setting* $B_j[H(d)] = 1, \forall 0 \le j \le i$ *with* $(i+1)$ *updates, MRB only sets* $B_i[H(d)] = 1$ *with single update*. Consequently, $B_0, \ldots, B_{i-1}$ lose a portion of its bits which are covered by B_i. Considering any bitmap B_i, it loses a portion of their bits which are covered by $B_{i+1}, \ldots B_{k-1}$ and will be recovered later when producing the cardinality estimate. The recording overhead per item is $2\mathcal{H}+\mathcal{A}$. For query, MRB checks the number of ones in B_i, denoted by U_i (excluding the bits covered by $B_{i+1}, \ldots B_{k-1}$), top-down, and find outs the first B_i whose $U_i \ge T$, where T is a predefined threshold. If no B_i is found, $i = 0$. B_i is considered as the bitmap with most suitable sampling probability. MRB collects the number of sampled distinct data items under the sampling probability p_i, which are recorded in $B_i, \ldots B_j, \ldots, B_{k-1}, \forall i \le j \le k-1$, each with a cardinality estimate of $-\frac{m}{k}\ln(1-\frac{U_j}{\frac{m}{k}})$. The query overhead is thus up to $m\mathcal{A}$. MRB calculates the sum of these cardinality estimates as $\sum_{j=i}^{k-1} -\frac{m}{k}\ln(1-\frac{U_j}{m/k})$, and divide it by p_i and produces the cardinality estimate $\hat{n}$ as

$$\hat{n} = 2^i \sum_{j=i}^{k-1} -\frac{m}{k}\ln(1-\frac{U_j}{m/k}) \tag{2}$$

The maximum estimate is produced when $i = k-1$ and $U_{k-1} = \frac{m}{k} - 1$, which is $2^{k-1}\frac{m}{k}\ln(\frac{m}{k})$. It is larger than the maximum estimate produced by m-bit bitmap ($m\ln m$) when $k > 2$. Although MRB achieves highest query throughput among the existing work, which will be validated in the experiments, it abandons the information in the bitmap under other sampling probabilities, making it less accurate. This motivates our design of SMB, which will be elaborated shortly.

Before we explain the following thread of cardinality estimators, we first define the geometric hash function.

Definition 1 (Geometric Hash Function): Function $G(x)$ is a geometric hash function of base $\frac{1}{2}$ if $G(x)$ is an integer and $G(x) = i$, $i \ge 0$, with the probability $2^{-(i+1)}$.
In practice, $G(x)$ can be performed by a uniform hash function $H(x)$, where $G(x) = \rho(H(x))$ and $\rho(y)$ is the number of leading zeros of y starting from the least significant digit.

FM, HLL++, HLL-TailC, etc. An FM register is a bitset F with b bits, with the ith bit denoted as $F[i]$, $0 \le i < b$. For any item d, we calculate $G(d)$ with $G(d) < b$ and set $F[G(d)] = 1$. Consider the cardinality estimate of FM. Roughly speaking, any item is hashed to the ith bit with probability $\frac{1}{2^{i+1}}$ and in turn, $F[i] = 1$ approximately represents 2^{i+1} of distinct data items in the data stream. Obviously, this probabilistic counting is too coarse, one-bit fluctuation resulting in a totally different estimate, especially for the significant bits. To this end, FM uses t registers, $F_0, \ldots, F_t$, to improve the estimation accuracy. For an m-bit FM, $t = \frac{m}{b}$ (suppose it is an integer) and any data item is mapped to $H(d) \pmod t$th register for recording. Totally, the recording overhead is $2\mathcal{H}+\mathcal{A}$. In practice, b is recommended to be 32 to accommodate a sufficient large estimation range. For query, FM traverses all the registers and produces the cardinality estimate as

$$\hat{n} = t \cdot 2^{\frac{\sum_{i=0}^{t-1} z_i}{t}} / \phi \tag{3}$$

where z_i is the number of consecutive ones of F_i starting from the least significant bit, and ϕ is pre-computed constant that is related to t. $\phi = 0.78$ when t is large enough. Refer to [1] for the value of ϕ under different t. The query overhead is $m\mathcal{A}$.

An HLL++ register Y is a counter of b bits (for estimation range up to 2^{2^b-1}), which represents an integer of range $0 \le Y \le 2^b - 1$. The value of b is recommended as 5. It is a compact version of the FM register. For any data item d, we calculate $G(d)$ with $G(d) \le 30$ and set $Y = \max\{Y, G(d)+1\}$. It has the same estimation accuracy issue as the FM register. Therefore, HLL++ also uses t registers. For an m-bit HLL++, $t = \frac{m}{5}$ (suppose it is an integer). The recording overhead per item is $2\mathcal{H}+5\mathcal{A}$. For query, HLL++ produces cardinality estimate through arithmetic mean $\bar{Y}$ of $Y_0, \ldots, Y_{t-1}$ as

$$\hat{n} = \alpha_t \cdot t \cdot \bar{Y} \ with\ \bar{Y} = t(\sum_{i=0}^{t-1} 2^{-Y_i})^{-1} \tag{4}$$

where α_t is a constant that can be calculated as $\alpha_t = 0.7213/(1+\frac{1.079}{t})$ when $t \ge 128$. For value of α_t under different t, refer to [20], [27]. Since all the registers are accessed, the query overhead is $m\mathcal{A}$. Furthermore, HLL++ corrects the estimation bias when the cardinality n is very small, up to $m = 5t$, using the solutions including the bitmap algorithm. In fact, HLL++ is not proposed at one stroke but belongs to a family of LogLog algorithms, including LogLog, SuperLogLog and HLL, which use the same data structure and recording operation but different estimation formulas. As a result, the recording and query overhead are the same as those of HLL++. We only describe HLL++ here as HLL++ is the state of the art in the family.

There are some optimizations based on HLL++. HLL-TailC reduces the size of each HLL++ register Y_i from

5 bits to 4 bits. The new register Y_i' stores the offset value $Y_i' = Y_i - \mathcal{B}$, where variable $\mathcal{B}$ is maintained and is equal to the $\min_{0 \le i < t} Y_i$. If Y' overflows, $Y_i' = 15$. For query, HLL-TailC recovers Y_i based on Y_i' and $\mathcal{B}$ and produces estimates with the formula for HLL++ in (4). More aggressively, HLL-TailC+ reduces size of each LogLog register Y_i from 5 bits to 3 bits at the cost of expensive query operations, which can only be done offline. Therefore, HLL-TailC rather HLL-TailC+ is considered in this paper. Refined HLL uses a different geometric hash function $G'(x)$, where $G'(x) = 0, 1, 2, 3, \ldots, j, \ldots$ with the probabilities of $\frac{1}{4}, \frac{1}{4}(\frac{3}{4})^1, \frac{1}{4}(\frac{3}{4})^2, \frac{1}{4}(\frac{3}{4})^3, \ldots, \frac{1}{4}(\frac{3}{4})^j, \ldots$, respectively. However, unlike HLL++ which has fixed coefficient α_t, Refined HLL needs to use a portion of the data stream to train a model and derive the coefficient, making it impractical for online cardinality estimation.

Usually, FM, HLL++ and HLL-TailC are equipped with hundreds or thousands of registers. Producing cardinality estimate needs to access all of them and calculate complex formula such as (4) for HLL++, incurring low query throughput.

MinCount, AKMV, etc. These solutions estimate stream cardinality using k-th minimum hash value of all data items. For each data item d, if we perform uniform hash $H(.) \in [0, 1]$. The expected distance between any neighboring hash values is $\frac{1}{n+1} \approx \frac{1}{n}$ by symmetry. Thus, the k-th minimum value v_k is expected to be $E(v_k) = \frac{k}{n}$ and we can estimate n by $\hat{n} = \frac{k}{v_k}$. When allocated m bits of memory, these solutions maintain $\frac{m}{32}$ (suppose it is integer) minimum hash values (suppose each hash value is 32 bit). The recording overhead for each data item is one hash and at least one comparison with the minimum hash values, i.e., $[32\mathcal{A}, m\mathcal{A}]$. The query overhead depends on the number of values these solutions access. AKMV access only v_k and lowers the estimation bias by revising the formula to $\hat{n} = \frac{k-1}{v_k}$. Its query overhead is thus $32\mathcal{A}$. However, it suffers from large variance as its estimate depends only on the v_k. To reduce the variance, MinCount divides the hash values into t buckets, each maintaining the k-th minimum value to reduce the estimation variance. The query overhead is thus $32t\mathcal{A} \in [32\mathcal{A}, m\mathcal{A}]$. Kane *et al.* [30] proposed a cardinality estimator which compresses $O(t \log \log n)$ bits of the LogLog family and $O(t \log n)$ bits of FM to $O(t)$ bits, under the same accuracy requirement. This from an information-theoretic point of view is a progress. However, this effort from a practical perspective is not successful as the actual memory consumption is not reduced. Moreover, the algorithm has a probability (up to $\frac{1}{32}$) to fail when recording and if so it will need to re-run the algorithm, making it impractical for online cardinality estimation.

Summary of existing cardinality estimators and our goal. Table I compares the recording and query overhead of each cardinality estimator. As this paper focuses on the online cardinality estimation, we highlight the query overhead. Among existing solutions for streaming data, MRB achieves lowest query overhead theoretically and experimentally (but still higher than SMB, see Section VI). But its estimation accuracy is not as high as HLL++. Moreover, an experimental survey [28] compared the estimation accuracy of LogLog, SuperLogLog, HLL, HLL++, MinCount, AKMV, under tens of datasets and drew the following conclusion. HLL++ is the best among the LogLog family. MinCount and ARMV perform worse than SuperLogLog, let along HLL and HLL+. Thus, HLL++ and its optimization work HLL-TailC (HLL-TailC is not evaluated in the survey) are the state of the art. However, as we have explained, they have high query overhead. Our goal is that SMB can simultaneously achieve the best estimation accuracy and the lowest query overhead (plus the lowest recording overhead), making it an efficient solution for online cardinality estimation.

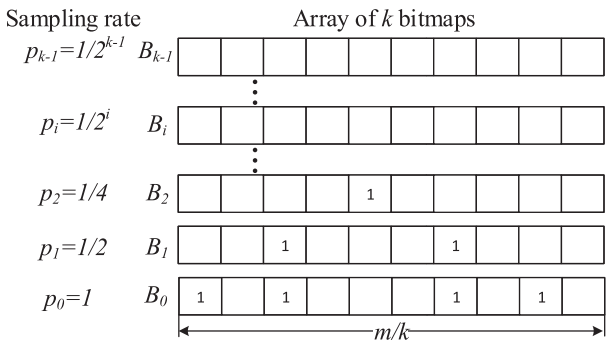


Fig. 1. Example of recording a small data stream in MRB. Only bits in B_0, B_1, B_2 are used to store the information. Memory space occupied by other bitmaps, i.e., $B_3, \ldots, B_{k-1}$ are wasted.

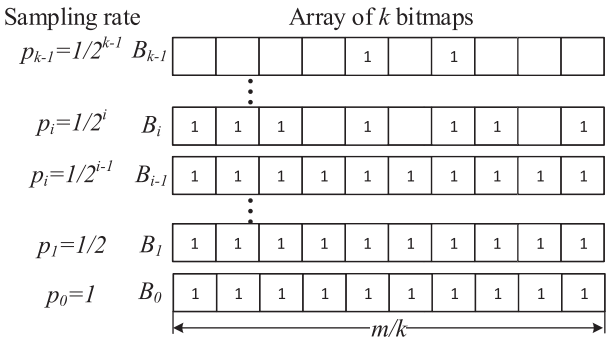


Fig. 2. Example of recording a large data stream in MRB. Bits in $B_0, \ldots, B_{i-1}$ are all set to ones, which cannot be used to produce cardinality estimate.

C. Other Related Work

Adaptive bitmap is a derived algorithm from MRB [3], [4] for stream cardinality estimation. Assuming that the stream cardinality in this interval is in the same order of magnitude as that in the previous one (measured by a small MRB), adaptive bitmap sets a suitable sampling probability p and applies p to bitmap for precise estimation. However, when the cardinality changes significantly from one interval to the subsequent one, the value of p will be improperly set and the cardinality estimate produced from bitmap will be ruined.

There is some work that designs a compact data structure (called sketch) for estimating cardinalities of multiple data

TABLE I

PERFORMANCE COMPARISON OF THE PROPOSED SMB ALGORITHM AND EXISTING SOLUTIONS WHEN EACH SOLUTION IS ASSIGNED m BITS AND THE STREAM CARDINALITY n IS UP TO THE MAGNITUDE OF 2^{32}. NOTE THAT $\mathcal{H}$ REPRESENTS THE OVERHEAD FOR ONE HASH OPERATION AND $\mathcal{A}$ REPRESENTS THE AVERAGE OVERHEAD OF ACCESSING 1-BIT MEMORY. WE USE THE NUMBER OF HASH OPERATIONS USED AND THE MEMORY ACCESSED PER DATA ITEM TO ROUGHLY SHOW THE RECORDING AND QUERY OVERHEAD. p IS THE SAMPLING PROBABILITY. THE ACCURACY RESULTS MOSTLY COME FROM THE EXPERIMENTAL SURVEY [28] ON A NUMBER OF DATASETS. NOTE THAT BITMAP IS ONLY ACCURATE IN ITS SMALL ESTIMATION RANGE

Solutions	Data Structure	Core Method	Recording Overhead	**Query Overhead**	Accuracy
Bitmap [2]	Array of m bits	Count number of 1s	$\mathcal{H}+\mathcal{A}$	$m\mathcal{A}$	High
MRB [3], [4]	32 $\lfloor\frac{m}{32}\rfloor$-bit bitmaps	Bitmap + sampling	$2\mathcal{H}+\mathcal{A}$	$[32\mathcal{A}, m\mathcal{A}]$	Medium
FM [1]	$\lfloor\frac{m}{32}\rfloor$ 32-bit registers	Count consecutive 1s	$2\mathcal{H}+\mathcal{A}$	$m\mathcal{A}$	Medium
LogLog [26]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	Count leading 0s	$2\mathcal{H}+5\mathcal{A}$	$m\mathcal{A}$	Medium
SuperLogLog [26]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	Count leading 0s	$2\mathcal{H}+5\mathcal{A}$	$m\mathcal{A}$	Medium
HLL [27]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	Count leading 0s	$2\mathcal{H}+5\mathcal{A}$	$m\mathcal{A}$	High
HLL++ [20]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	Count leading 0s	$2\mathcal{H}+5\mathcal{A}$	$m\mathcal{A}$	High
Refined HLL [6]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	Count leading 0s	$2\mathcal{H}+5\mathcal{A}$	$m\mathcal{A}$ + train overhead	High
HLL-TailC [30], [31]	$\lfloor\frac{m}{4}\rfloor$ 4-bit registers	Count leading 0s	$2\mathcal{H}+4\mathcal{A}$	$m\mathcal{A}$	High
HLL-TailC+ [30], [31]	$\lfloor\frac{m}{3}\rfloor$ 3-bit registers	Count leading 0s	$2\mathcal{H}+3\mathcal{A}$	Offline	High
Kane et al. [32]	$\lfloor\frac{m}{5}\rfloor$ 5-bit registers	k-th minimum value	$\geq(8\mathcal{H}+39*5\mathcal{A})$	$32\mathcal{A}$	Medium
MinCount [24]	$\lfloor\frac{m}{32}\rfloor$ 32-bit registers	k-th minimum value	$[\mathcal{H}+32\mathcal{A}, \mathcal{H}+32m\mathcal{A}]$	$[32\mathcal{A}, m\mathcal{A}]$	Medium
AKMV [25]	$\lfloor\frac{m}{32}\rfloor$ 32-bit registers	k-th minimum value	$[\mathcal{H}+32\mathcal{A}, \mathcal{H}+32m\mathcal{A}]$	$32\mathcal{A}$	Low
SMB	Array of m bits	Bitmap+ sampling	$(1+p)\mathcal{H}+p\mathcal{A}$	$32\mathcal{A}$	High

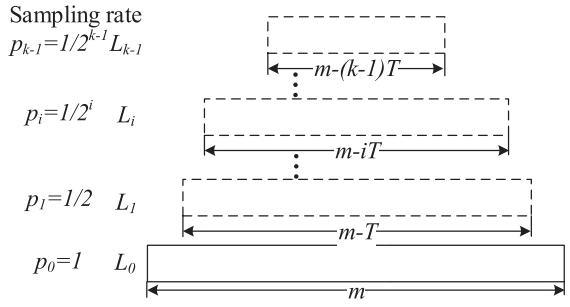


Fig. 3. Illustration of the recording process in SMB. Initially, data items are recorded in L_0 (bitmap with m bits) with a sampling probability $p_0 = 1$. When T bits are set to one, the remaining $m-T$ bits of zeros form a logical bitmap L_1, and the sampling probability changes to $p_1 = \frac{1}{2}$. This morphing process is imaginary and the process repeats each time the number of ones in the logical bitmap reaches the threshold T.

streams [10], [11], [33]–[35]. These sketches all use the cardinality estimators, e.g., bitmap, FM, HLL, and MRB, as plug-ins, and allow different data streams to be recorded in the same cardinality estimators. For instance, OpenSketch [33] uses multiple arrays of bitmaps; bSketch [10] is a generalized sketch framework that can be plugged in bitmap, FM, and HLL; SpreadSketch [34] uses MRB to identify superspreaders. They usually need to record/query each data item in multiple cardinality estimators to reduce the estimation error caused by the sharing of cardinality estimators among data streams. Therefore, they benefit from the improvement of estimation accuracy, memory efficiency, and high recording/query throughput of the internal plug-in cardinality estimators. We stress that SMB can also act as plug-in for these sketches and the performance improvement by our work can benefit these sketches accordingly. There is also other works on cardinality estimation for multiple data streams, including, AROMA [36], CSE [37], vHLL [11], VF [9].

III. SELF-MORPHING BITMAP

In this section, we first explain our motivation for SMB. After that, we present the design of SMB and prove its properties.

A. Motivation

It is well known that the Bitmap estimator [2], [37] has a limited estimation range of $-m \ln m$ [2], which is a serious problem in practice for large date streams [11], whereas FM, HLL++ and MRB have practically unlimited ranges by extending their register sizes or using smaller sampling probabilities.

As our experimental results will show, MRB achieves better recording throughput and query throughput than FM and HLL++, thanks to its simpler recording/querying operations. In terms of estimation accuracy, HLL++ is better than MRB and FM in most configurations. Overall, MRB achieves a superior balance between throughput and accuracy.

This paper proposes a new design that achieves much better recording throughput and query throughput than MRB and in the meantime better accuracy than HLL++. To motivate for our design, we examine MRB more closely and argue that it does not fully utilize its memory space.

• For a small data stream, it is likely that MRB only uses one or a few of its bitmaps (those with large sampling probabilities such as B_0). The memory space occupied by other bitmaps (with small sampling probabilities) will be wasted if no element is sampled for them and none of their bits is set to one, as illustrated by the example in Figure 1. In this case, had we used a single bitmap of all m bits, we would improve estimation accuracy.

• For a large data stream, according to (2), some of its bitmaps, B_0 through B_{i-1}, are not used because too many bits in them are set to ones, as illustrated by the example in Figure 2 and such saturated bitmaps result in large estimation error [2], [11]. To utilize B_0 through B_{i-1}, we have to

somehow expand these bitmaps so that they are not saturated with ones, yet we cannot increase the total number m of bits, nor should we reduce k, which would reduce the estimation range.

Moreover, after MRB determines that B_i has the most appropriate sampling probability p_i for this stream, it uses B_i through B_{k-1} in its estimation formula (2). However, B_{i+1} through B_{k-1} have used much smaller sampling probabilities than p_i, causing significant estimation error. We would achieve much better accuracy, had we applied p_i to B_{i+1} through B_{k-1} as well, which is however against the design structure of MRB.

Therefore, we need a new design to address the above accuracy-related issues. In addition, we want the new design to greatly improve recording throughput by reducing the overhead of item recording and improve query throughput by reducing the overhead of cardinality estimation. To achieve all these goals, our idea is *self-morphing bitmap* (SMB), which begins with a single bitmap L_0 of all m bits and continuously morphs itself through a series of steps as needed, with a sampling probability that decreases as more and more items are recorded.

We do not know the cardinality of the data stream beforehand. Therefore, we begin with a sampling probability $p_0 = 1$ for L_0. However, if the data stream turns out to be too big for p_0, which is indicated as the number of bits set to ones in L_0 reaches a threshold T, we need to adaptively reduce the sampling probability one notch down to $p_1 = \frac{1}{2}$ and *logically* prepare a new bitmap for sampling the rest of the stream by p_1. To do so, we first use formula (1) to estimate the number of distinct elements that have been recorded so far in L_0, and then conceptually morph L_0 to a new logical bitmap L_1 by removing the bits of ones. Morphing (i.e., removal of some bits) is an "imaginary" operation and does not result in any physical overhead. We simply treat the bits of zeros in L_0 as a logical bitmap L_1 and our cardinality estimation formula will account for the impact of conceptually removing the bits of ones.

The above process repeats, as illustrated in Fig. 3: $\forall 0 \leq i < k$, if we find the number of bits set to ones in L_i reaches the threshold T, we will reduce the sampling probability from $p_i = (\frac{1}{2})^i$ to $p_{i+1} = (\frac{1}{2})^{i+1}$. We then estimate the number of distinct elements having been recorded in L_i, and finally treat the bits of zeros in L_i as a new logical bitmap L_{i+1} to record the remaining data stream. Our evaluation shows that this design of SMB achieves an estimation accuracy better than MRB, FM and HLL++.

Next we explain intuitively how SMB achieves an average recording throughput much higher than MRB (which is in turn higher than FM and HLL++). The design of SMB ensures that at any time there is only one bitmap L_i and one sampling probability p_i under operation. The fraction of arrival items that will be sampled for recording is equal to p_i. As an example, if $p_i = (\frac{1}{2})^8$, only one out of every 256 data items is recorded on average. The recording overhead is amortized over many items, which reduces the average overhead per item. In contract, MRB operates k bitmaps at any time, which together record a fraction p_0 of all data items, incurring higher overhead per item because p_0 is typically set to one.

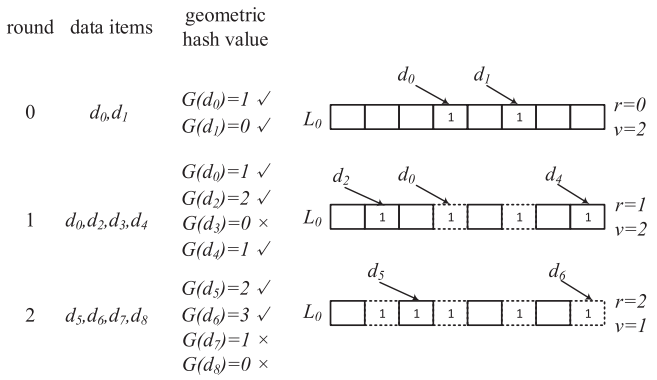


Fig. 4. Example for the recording operation of SMB. Note that in each round r with $r > 0$, the logical bitmap L_r consists of bits of zeros in L_0 (all the rectangles with solid lines).

The recording throughput of SMB changes over time since the sampling probability changes. In practice, when we have to handle many small/large data streams together with different arrival times, we shall allocate one self-morphing bitmap for each data stream, with independently-changing sampling probability. While the recording throughput of individual streams increases over time, the aggregate recording throughput of all streams is more stable as new streams arrive and existing streams terminate.

The proposed self-morphing bitmap also has a higher query throughput than MRB, FM, and HLL++ because its computation is simpler than (2), (3), and (4). When a query on stream spread is made, we only need to calculate (7) for the current bitmap and add the result to what the previous logical bitmaps have recorded (which does not change over time and is thus computed before we morph into the current bitmap).

The detailed design of self-morphing bitmap is given next.

B. Self-Morphing Bitmap Design

SMB maintains one bitmap L_0 with length of m. The recording process consists of a series of rounds identified by a *round index* r, which starts from zero and increases by one each time after T bits are set to ones, where T is a pre-specified threshold value. The ith round will have a sampling probability $p_r = \frac{1}{2^r}$. In the beginning, $r = 0$ and $p_0 = 1$. We use a variable v to keep track of the number of bits that are set from zeros to ones set in the current round. When v reaches T, we reset $v = 0$ and increase r by one to start the next round.

Recording: Upon the arrival of an item d. We perform a geometric hash operation $G(d)$ (see Definition 1), and do the following three steps.

Step 1: If $G(d) \geq r$, go to next step; otherwise, ignore the item. This step samples the item with the probability of 2^{-r}, which we will prove shortly. Since $r = 0$ initially and r will only increase during recording, the sampling probability in this step will decrease from 1 to $\frac{1}{2^1}$, $\frac{1}{2^2}$ …

Step 2: Perform uniform hash operation $H(d) \in [0, m-1]$. If $L_0[H(d)] = 0$, set $L_0[H(d)] = 1$, increment v by 1, and proceed to next step. Otherwise, do nothing.

Step 3: If $v \geq T$, we increment r by 1 and update $v = 0$; otherwise, do nothing. The threshold $T \leq m$ is a predefined parameter and we will theoretically give the optimal setting. When the number of ones exceeds the threshold, it means the cardinality of the data stream is large enough and we should adjust the sampling probability to be smaller. We will prove that each item can only be recorded by its first appearance and its subsequent appearances will be ignored in next section. We give the following example to facilitate understanding of the recording process.

Example: We give an example shown in Figure 4 to illustrate the recording operation of SMB, where $m = 8$, $T = 2$, and the data stream $D = \{d_0, d_1, d_0, d_2, d_3, d_4, d_5, d_6, d_7, d_8\}$. Initially $r = 0$, $v = 0$, and all bits in B are set as zeros. SMB will process incoming items one by one and experience the following three rounds of recording.

Round 0: for items d_0 and d_1, we first calculate their geometric hash values, $G(d_0)$, $G(d_1)$. Since $G(d_0) = 1 \geq r = 0$ and $G(d_1) = 0 \geq r = 0$, d_0 and d_1 get sampled in Step 1 and proceed to Step 2. Let $H(d_0) = 3$ and $H(d_1) = 5$. SMB sets $L_0[3] = 1$ and $L_0[5] = 1$. Accordingly, v is incremented by one twice, i.e., $v = 2$. Since $v \geq T$, we proceed to round 1 and update $r = 1$ and $v = 0$,.

Round 1: The logical bitmap L_1 in round 1 consists of all bits in L_0 excluding $L_0[3]$ and $L_0[5]$. As shown in the figure, L_1 contains all rectangles with solid lines in L_0. For the subsequent items d_0, d_2, d_3, d_4. d_0 gets sampled by Step 1 as $G(d_0) = 1 \geq r = 1$. In Step 2, since $L_0[H(d_0)] = 1$, we do nothing. Next item d_2 is sampled in Step 1 as $G(d_2) = 2 \geq r = 1$, and we set $L_0[H(d_2)] = L_0[1] = 1$ and $v = 1$ in Step 2. d_3 is dropped as $G(d_3) = 0 < r = 1$. As $G(d_4) = 1 \geq r = 1$, we set $L_0[H(d_4)] = L_0[7] = 1$ and $v = 2$. Since $v \geq T$, we update $r = 2$ and $v = 0$, and go to round 2.

Round 2: We morph L_1 to L_2 by treating all bits of zeros in L_1 as a new bitmap (rectangles with solid lines in round 2). For the subsequent items d_5, d_6, d_7, d_8. As $G(d_5) = 2 \geq r = 2$ and $L_0[H(d_5)] = L_0[2] = 0$, we set $L_0[H(d_5)] = 1$, and update $v = 1$. The next data item d_6 is dropped in Step 2 as $G(d_6) \geq r = 2$ and $L_0[H(d_6)] = L_0[7] = 1$. d_7 and d_8 are dropped as they do not pass Step 1 ($G(d_7) = 1 < r = 2$ and $G(d_8) = 0 < r$).

Querying: After the measurement period, the integer r indicates SMB experiences $r+1$ rounds of recording. Consider arbitrary round $i, 0 \leq i \leq r$. Step 1 of the recording operation samples each data item with probability p_i. We give the following lemma to prove $p_i = \frac{1}{2^i}, 0 \leq i \leq r$.

Lemma 1: For the ith round of recording with $0 \leq i \leq r$, we have $p_i = 2^{-i}$.

The proof can be easily derived and is thus omitted. After passing Step 1, data items will be recorded in the (logical) bitmap L_i with m_i bits, where $m_i = m - iT$. Let U_i be the number of ones set in L_i, we have

$$U_i = \begin{cases} T, & 0 \leq i < r \\ v, & i = r \end{cases} \tag{5}$$

Consider the estimate n_i produced by L_i in round i. By applying the estimation formula of bitmap [2], we have

$$\hat{n}_i = -m_i \ln(1 - U_i/m_i) \tag{6}$$

From (5) and (6), we have

$$\begin{aligned} \hat{n}_i &= -m_i \ln(1 - T/m_i) \quad 0 \leq i < r \\ \hat{n}_r &= -m_r \ln(1 - U_r/m_r) \end{aligned}$$

For round i with $i > 0$, bitmap L_i is logical and all its m_i bits come from m bits in L_0. Due to the uniformness of the hash value, only m_i/m items will be hashed to L_i. This will enlarge the estimate n_i by m/m_i folds. Therefore, the total number of distinct items recorded by the logical bitmaps, L_0 through L_{r-1}, after sampling is accounted for, is

$$\begin{aligned} S_r &= \sum_{i=0}^{r-1} \frac{1}{p_i} \frac{m}{m_i} \hat{n}_i = \sum_{i=0}^{r-1} \frac{1}{p_i} \frac{m}{m_i} \hat{n}_i \\ &= \sum_{i=0}^{r-1} \frac{1}{p_i} \frac{m}{m_i} - m_i \ln(1 - T/m_i) \\ &= \sum_{i=0}^{r-1} 2^i - m \ln(1 - T/m_i) \end{aligned}$$

Because the value of S_r does not change during the rth round (i.e., the current round), we can cache its value for use. For a special case of $r = 0$, $S_r = 0$. To answer a query on the current cardinality estimate $\hat{n}$, we only need to compute estimate in the current round, i.e.,

$$\frac{1}{p_r} \frac{m}{m_r} \hat{n}_r \tag{7}$$

and add it to S_r.

$$\begin{aligned} \hat{n} &= S_r + \frac{1}{p_r} \frac{m}{m_r} \\ \hat{n}_r &= S_r - 2^r m \ln(1 - \frac{v}{m - rT}) \end{aligned} \tag{8}$$

Note that under the same memory allocation, if the length of each bitmap in MRB is T, SMB's maximum estimate is larger than that of MRB. It can be proved by only considering the maximum estimate produced in the last round of SMB ($\hat{n}_r$). The maximum estimate of $\hat{n}_r$ is produced when $r = \frac{m}{T} - 1 = k - 1$ (suppose it is integer) and $v = m - rT - 1 = T - 1$, which is $2^{k-1} m \ln(T)$ and is larger than the maximum estimate produced by MRB ($2^{k-1} \frac{m}{k} \ln(\frac{m}{k}) = 2^{k-1} T \ln(T)$).

IV. Generalized Self-Morphing Bitmap

The design of SMB in the above section employs a series of specific sampling probabilities $\{p_0, p_1, p_2, \ldots\} = \{1, \frac{1}{2}, \frac{1}{2^2}, \ldots\}$, where the sampling probability decreases to a portion $p = \frac{1}{2}$ of the previous one every time we proceed to the next round of recording. However, our experiments under datasets with different stream cardinalities and various memory allocation in Section VI reveal that setting $p = \frac{1}{2}$ does not always promise the most accurate estimation. This motivates us to *generalize* the design of SMB by allowing p to be any constant in range of $(0, 1)$.

The generalized SMB also maintains one bitmap L_0 with length of m. It inherits the terminologies, i.e., *rounds, threshold,* and notations from the SMB in the previous section. The difference is that the sampling probability for the ith round of recording (in Step 1) is $p_i = p^i$ with $p \in (0,1)$. In the beginning, $r = 0$, $v = 0$ and $p_0 = 1$. To implement the sampling process for arbitrary p, we cannot employ the geometric hash function $G(\cdot)$ in Definition 1 that is specially designed for the case where $p = \frac{1}{2}$. Instead, we defined the generalized geometric hash function as follows.

Definition 2: [Generalized geometric hash function] Function $G^*(x)$ is a generalized geometric hash function of for arbitrary base $p \in (0,1)$ if $G^*(x)$ outputs integer $i \geq 0$ with the following probability

$$G^*(x) = \begin{cases} 0, & \text{with probability } 1-p \\ i \geq 1, & \text{with probability } (1-p)p^i. \end{cases} \quad (9)$$

The above definition guarantees the following property of $G^*(x)$.

Lemma 2: For any constant $p \in (0,1)$ and integer $i \geq 0$, we have

$$\Pr(G^*(x) \geq i) = p^i$$
$$\sum_{i=0}^{\infty} \Pr(G^*(x) = i) = 1 \quad (10)$$

The above lemma can be easily proved and is thus omitted. In practice, $G^*(x)$ can be implemented by using a uniform hash function $H^*(x)$ whose output range is $[0,1)$. $G^*(x)$=i if $H^*(x) \in [1-p^i, 1-p^i+(1-p)p^i)$=$[1-p^i, 1-p^{i+1})$.

Recording: Upon the arrival of an item d. We perform a generalized geometric hash operation $G^*(d)$ (see Definition 2), and do the following three steps.

Step 1: If $G^*(d) \geq r$, go to next step; otherwise, ignore the item. This step samples the item with the probability of $p_r = \Pr(G^*(d) \geq r) = p^r$, which is validated by Lemma 2. Since $r = 0$ initially and r will only increase during recording, the sampling probability in this step will decrease from 1 to p, p^2 …for any $p \in (0,1)$.

Steps 2 and 3 are the same as those of SMB in the previous section.

The recording operation of SMB is described in Algorithm 1. We will prove that each item can only be recorded by its first appearance. Its subsequent appearances will be ignored.

Querying: After the measurement period, the integer r indicates SMB experiences $r+1$ rounds of recording. Consider arbitrary round $i, 0 \leq i \leq r$. Step 1 of the recording operation samples each data item with probability $p_i = p^i$. After passing Step 1, data items will be recorded in the (logical) bitmap L_i with m_i bits, where $m_i = m - iT$. Since Steps 2 and 3 are the same as those in the previous section, we directly employ the formula in (6) for n_i that represents the number of distinct items recorded in L_i of the ith round. After the sampling is accounted for, we can estimate the number of distinct items recorded in the logical bitmaps, L_0 through L_{r-1} as

$$S_{p,r} = \sum_{i=0}^{r-1} \frac{\hat{n}_i}{p_i}\frac{m}{m_i} = \sum_{i=0}^{r-1} -\frac{1}{p^i} m \ln(1 - \frac{T}{m_i}). \quad (11)$$

Algorithm 1 Recording a Data Item in the Generalized SMB

```
1: Input: data item d, T
2: Action: record d, and update r and v
3: if G*(d) ≥ r then
4:   if L0[H(d)] = 0 then
5:     L0[H(d)] = 1
6:     v = v + 1
7:     if v ≥ T then
8:       r = r + 1
9:       v = 0
10:    end if
11:  end if
12: end if
```

Algorithm 2 Querying on the Generalized SMB

```
1: Input: T, S_p
2: Output: the cardinality estimate of the data stream
3: return S_p[r] − (1/p^r) m ln(1 − v/(m − rT))
```

Given p, $S_{p,r}$ does not change during the rth round, i.e., the current round, we can pre-compute it and cache its value for use. For the special case of $r = 0$, $S_{p,r} = 0$. To answer a query on the current cardinality estimate $\hat{n}$, we only need to compute estimate in the current round and add it to $S_{p,r}$.

$$\hat{n} = S_{p,r} + \frac{\hat{n}_r}{p_r}\frac{m}{m_r} = S_{p,r} - \frac{1}{p^r} m \ln(1 - \frac{v}{m - rT}) \quad (12)$$

The query operation is described in Algorithm 2. Comparing with the existing work HLL++ and HLL-TailC with query overhead of $m\mathcal{A}$, SMB only accesses two integer values, r and v. For a stream cardinality in the magnitude of 2^{32}. r is at most 32 and can be assigned 6 bits. v is at most T and 26 bits can make v up to $2^{26} - 1$, which is enough. Totally, the query overhead is $32\mathcal{A}$. Comparing with MRB which only uses the information in the bitmap with the best sampling probability, SMB utilizes all the recorded information (see (8)), making it more accurate. Our experiment will show that SMB achieves the best performance in estimation accuracy, recording throughput and query throughput, compared to existing state-of-the-art solutions.

Theorem 3: For any data item d, its first appearance may be recorded by SMB. But its subsequent appearances will be blocked.

Proof: We prove by contradiction. Let d' and d'' be the data item d of the first appearance and the second appearance, respectively. Recall that the generalized geometric hash function $G^*(\cdot)$ is built based on the uniform hash function. Due to the pseudo-randomness of the hash function, we have $G^*(d') = G^*(d'') = G^*(d)$. Assume that when processing d'', there exist $G^*(r) \geq r$ and $L_0[H(d'')] = 0$. Since d' appears before d'' and the value of r will only increase. Therefore, we know $G^*(d') \geq r$ when recording d' and d' will pass Step 1. In Step 2, SMB will set $L_0[H(d')] = 1$, which contradicts with the assumption that $L_0[H(d'')] = 0$ as $H(d') = H(d) = H(d'')$. Thus, the theorem holds. □

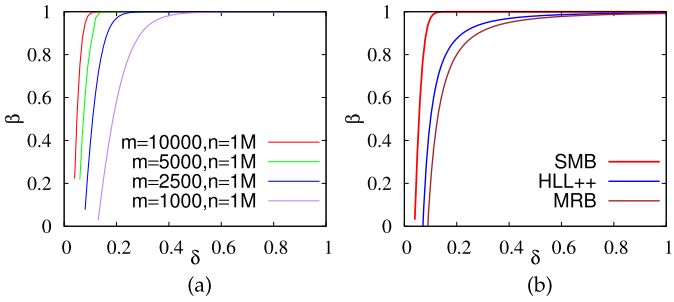


Fig. 5. (a) β w.r.t. δ for SMB when $n = 1$M, under memory allocations of 1000 bits, 2500 bits, 5000 bits and 10000 bits, respectively. (b) β w.r.t. δ for SMB and MRB when $n = 1$M and $m = 10000$bits.

V. Analysis and Parameter Setting

This section first presents a theorem to show the theoretical estimation performance of SMB and interprets the theorem. After that, we present the optimal setting for parameter T.

A. Estimation Error Bound

Theorem 4: Let n and $\hat{n}$ be the actual cardinality and estimated cardinality, respectively. The probability that the relative error $|\frac{n-\hat{n}}{n}|$ is bounded by an arbitrary constant $\delta \in (0, 1)$ must be larger than β, i.e.,

$$\Pr(|\frac{n-\hat{n}}{n}| \leq \delta) \geq \beta = 1 - 2e^{-\frac{(m_r - U_r + 1)p^r}{m}\frac{\delta^2 n(1-\delta)}{2}}, \quad (13)$$

where r is the maximum integer value that satisfies $n(1+\delta) \geq S_{p,r}$ and $U_r \leq T$ is the maximum integer value that satisfies $n(1+\delta) \geq S_{p,r} + \frac{m}{p^r}(-\ln\frac{m_r - U_r}{m_r})$ (but not larger than T) and the value of array S_{p_r} can be found in (11).

Proof: Let X_i^j be the number of distinct data items processed (including the data items that are not sampled by Step 1) to make U_i increase to j from $j-1$ in the ith round of recording. When $U_i = j-1$ there are $m_i - (j-1)$ bits in L that are zero. Due to the uniformness of the hash value, X_i^j is a geometric random variable and we have

$$E(X_i^j) = \frac{m}{(m_i - j + 1)p_i} = \frac{m}{(m_i - j + 1)p^i}. \quad (14)$$

Totally, there are rT variables from the first r rounds of recording, i.e., $X_0^1, X_0^2, \ldots, X_0^T, X_1^1, \ldots, X_{r-1}^1, \ldots, X_{r-1}^T$, and U_r variables in the rth round of recording, i.e., $X_r^1, X_r^2, \ldots, X_r^{U_r}$. These variables are mutually independent. Denote

$$X = \sum_{0 \leq i < r, 1 \leq j \leq T} X_i^j + \sum_{1 \leq j \leq U_r} X_r^j.$$

The recording process of the data item terminates after the event $X_r^{U_r}$ happens and before the event $X_r^{U_r+1}$ happens. Therefore, the actual stream cardinality locates between X and $X + X_r^{U_r}$. We call the data stream whose recording terminates when its last distinct data item exactly set a bit with value of zero to one in L as the integer stream and other streams as non-integer stream. We will show with the same U_r and r, worst case happens to the integer stream. By considering the worse case, we have

$$X = n, \quad (15)$$

and the following property.

Lemma 5: $\lim_{m\to\infty} \mathrm{E}(X) = \hat{n}$.

Proof: Consider the ith round of recording with $0 \leq i < r$. From (14), we know

$$\mathrm{E}(\sum_{1 \leq j \leq T} X_i^j) = \sum_{1 \leq j \leq T} \frac{m}{(m_i - j + 1)p^i} = \frac{m}{p^i}(H_{m_i} - H_{m_i - T})$$

Note that H_x is the x-th harmonic number. Utilizing the asymptotics of the harmonic numbers [38], we obtain

$$\begin{aligned}\mathrm{E}(\sum_{1 \leq j \leq T} X_i^j) &= \frac{m}{p^i}(H_{m_i} - H_{m_i - T}) \\ &= \frac{m}{p^i}(\ln m_i - \ln(m_i - T)) \\ &= \frac{m}{p^i}\ln(m_i/(m_i - T)) = \frac{m}{m_i p^i}\hat{n}_i\end{aligned}$$

Similarly, for the rth round of recording, we have $\mathrm{E}(\sum_{1 \leq j \leq U_r} X_r^j) = \frac{m}{m_r p^r}\hat{n}_r$. Since the estimate in each round of recording is equal to the actual expected number of distinct data items processed in that round. We have $\lim_{m\to\infty} \mathrm{E}(X) = \hat{n}$. □

Employing the upper bound for the sum of the geometric random variables (Theorem 2.1 for the upper tail and Theorem 3.2 for the lower tail in [39]), we have

$$\begin{aligned}\Pr(X \geq (1+\delta)\mathrm{E}(X)) &\leq e^{-\frac{(m_r - U_r + 1)}{2^r m}\mathrm{E}(X)(\delta - \ln(1+\delta))} \\ \Pr(X \leq (1-\delta)\mathrm{E}(X)) &\leq e^{-\frac{(m_r - U_r + 1)}{2^r m}\mathrm{E}(X)(-\delta - \ln(1-\delta))},\end{aligned}$$

where $\delta \in (0, 1)$ and variable p_* in paper [39] represents the minimum success probability of all geometric variables, which is $\frac{(m_r - U_r + 1)p^r}{m}$ in the context of this proof. Usually, we bound X by a small δ and hence we have $\ln(1+\delta) = \delta + \delta^2/2 + o(\delta^2)$ and $\ln(1-\delta) = -\delta + \delta^2/2 + o(\delta^2)$. Therefore, the above equations can be combined together as

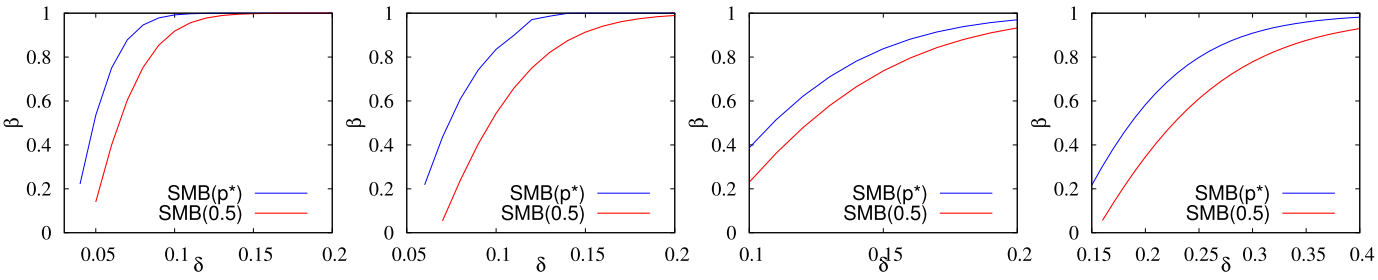
$$\Pr(|X - \mathrm{E}(X)| \geq \delta\mathrm{E}(X)) \leq 2e^{-\frac{(m_r - U_r + 1)p^r}{m}\mathrm{E}(X)\frac{\delta^2}{2}}. \quad (16)$$

Since the number of bits in L_i is usually sufficiently large, e.g., 10^4, we have $E(X) = \hat{n}$ (see Lemma 5). From (15), (16) can be rewritten as

$$\begin{aligned}&\Pr(|n - \hat{n}| \geq \delta\hat{n}) \leq 2e^{-\frac{(m_r - U_r + 1)}{2^r m}\hat{n}\frac{\delta^2}{2}} \\ \Leftrightarrow\ &\Pr(\frac{|n-\hat{n}|}{n} \geq \delta) \leq 2e^{-\frac{(m_r - U_r + 1)p^r}{m}\frac{\delta^2 n^2}{2\hat{n}}} \\ \Leftrightarrow\ &\Pr(\frac{|n-\hat{n}|}{n} \geq \delta) \leq 2e^{-\frac{(m_r - U_r + 1)p^r}{m}\frac{\delta^2 n(1-\delta)}{2}} \\ \Leftrightarrow\ &\Pr(\frac{|n-\hat{n}|}{n} \leq \delta) \leq 1 - 2e^{-\frac{(m_r - U_r + 1)p^r}{m}\frac{\delta^2 n(1-\delta)}{2}} \quad (17)\end{aligned}$$

The last inequality shows that under the same value of U_r and r, right part for the integer stream (with smaller n) is smaller than that for the non-integer stream (with larger n). This answers the previous argument that worst case happens to the integer stream when U_r and r are the same.

The right part of the last inequality decreases as r and U_r increases. Consider the maximum value of r. In this case $\hat{n} = n(1+\delta)$. We have $n(1+\delta) = \hat{n} \geq S_{p,r}$, where r is upper-bounded by the maximum integer value that makes the inequality hold.



(a) $m = 10000$, $n = 1\text{M}$ (b) $m = 5000$, $n = 1\text{M}$ (c) $m = 2500$, $n = 1\text{M}$ (d) $m = 1000$, $n = 1\text{M}$

Fig. 6. Comparison between the generalized SMB under $p = p^*$, denoted as SMB(p^*) and the SMB under $p = 0.5$ (designed in Section III and denoted as SMB(0.5)), under different values of m and $n = 1\text{M}$. p^* is the optimal setting for p and its values under given m and n are presented in Table II. All the plots show that the generalized SMB can bound the estimation error by arbitrary δ with higher probability than SMB(0.5).

Consider the maximum value of U_r with $U_r \leq T$. Since $\hat{n} \leq (1+\delta)n$, we can have the upper bound for U_r. $n(1+\delta) \geq \hat{n} \geq S_{p,r} + \frac{m}{p^r}(-\ln \frac{m_r - U_r}{m_r})$. □

Next, we interpret Theorem 4. According to (13), the error bound is closely related to variable p, m, T and n. For ease of understanding the error bound, we also give visual display by plotting β with respect to δ when $n = 1\text{M}$. We choose four representative values of m, i.e., $m = 10\text{k}$, 5k, 2.5k and 1k with the unit of bit and T, p are optimally set. The setting of T and p will be explained shortly in the following subsection. The error bound is shown in Figure 5(a). As we can see, the estimation error is bounded by small δ with high probability. For instance, when $m = 10000$ bits and δ =0.1, $\beta = 0.992$. That means $\frac{|n-\hat{n}|}{n} < 0.1$ happens with the probability of $\geq 99.2\%$. Even when m is small, i.e., 1000, $\frac{|n-\hat{n}|}{n} < 0.30$ happens with the probability of $\geq 90.8\%$. Note that the results in Figure 5(a) act as an example for n. When n varies, similar figures can be drawn.

We compare SMB with MRB and HLL++ in terms of the theoretical estimation error bound. The values of p and T are optimally set. For a fair comparison, the length of each bitmap in MRB is set as T. In the original paper for MRB [3], [4] and for HLL++ [20], [27], the standard error $|\frac{n-\hat{n}}{n}|$ is given, from which we can bound the relative error by δ with a probability β using Chebyshev's inequality. We plot β with respect to δ for SMB, MRB and HLL++ when $n = 1M$ and $m = 10000$ for each algorithm in Figure 5(b), which shows under the same δ, SMB's β is larger than that of MRB and HLL++. That means SMB is more likely to bound the estimation error with an arbitrary constant δ than MRB and HLL++.

B. Parameter Setting for T and p

We first discuss how to set the value of T when p is given. The SMB can support maximum $\frac{m}{T}$ rounds of recording, which should be larger than or equal to $r+1$, i.e., $\frac{m}{T} \geq r+1$. This constraint gives the upper bound of T. We consider the optimal integer value of m/T, which should be large enough to accommodate the stream cardinality and meanwhile makes β maximized. Given a fixed p, we can always derive the optimal value of T using numerical computing under the values of m and n. By brute-force testing all the values of p with a granularity of 1%, we can find out the optimal value of p that maximizes β, denoted as p^*. SMB under $p = p^*$ can bound the estimation error with the highest probability. We provide the optimal setting of p, T under difference values of m and n in Table II

TABLE II

PARAMETER SETTING OF GENERALIZED SMB: OPTIMAL VALUE OF p (DENOTED AS p^*), NUMBER OF BITMAPS k AND LENGTH FOR EACH BITMAP m/k UNDER GIVEN n, m

n \ m	10k			5k			2.5k			1k		
-	p^*	m/k	k	p^*	m/k	k	p^*	m/k	k	p^*	m/k	k
1M	0.40	1000	10	0.53	416	12	0.42	277	9	0.42	76	13
900k	0.43	1111	9	0.44	500	10	0.43	208	12	0.48	76	13
800k	0.41	1428	7	0.45	500	10	0.50	192	13	0.40	90	11
700k	0.42	1428	7	0.46	500	10	0.41	250	10	0.48	71	14
600k	0.44	1000	10	0.41	555	9	0.42	250	10	0.40	83	12
500k	0.41	1250	8	0.40	500	10	0.53	208	12	0.41	83	12
400k	0.43	1250	8	0.41	714	7	0.45	250	10	0.44	90	11
300k	0.42	1666	6	0.44	500	10	0.47	250	10	0.44	83	12
200k	0.47	1666	6	0.43	625	8	0.41	357	7	0.51	76	13
100k	0.45	2000	5	0.47	833	6	0.43	312	8	0.40	100	10
80k	0.49	2000	5	0.50	625	8	0.41	416	6	0.44	111	9

In practical settings, when there is no knowledge of the real cardinality of the data stream or we expect to assign identical T for a number of data streams with different cardinalities, we can choose the parameter setting of T under a large n (that is safe enough to accommodate the data stream) or the maximum streaming cardinality n_m among all data streams. It is because the optimal setting for $n = n_m$ can also be applied for the case where $n \in [0, n_m]$. From (13), we know β is affected by $\frac{(m_r - U_r + 1)n}{2^r m}$, where $\frac{2^r m}{(m_r - U_r + 1)}$ represents the expected number of distinct data items required make v increases from $U_r - 1$ to U_r. The ratio is $O(\frac{1}{T})$ and always stays very large, meaning that β is guaranteed under when n varies.

C. Theoretical Advantage of the Generalized SMB

We compare SMB under $p = p^*$ with SMB under $p = 0.5$, denoted as SMB(p^*) and SMB(0.5), respectively. Setting $m = 10000, 5000, 2500, 1000$, and $n = 1\text{M}$, we plot the value of β with respect to δ in Figure 6. As we can see, SMB(p^*) always bound the estimation error by the same δ

with a higher probability. For instance, when $m = 10000$, $n = 1$M, $\delta = 0.1$, β of SMB(p^*) is 99.2% and that of SMB(0.5) is 91.8%. When $m = 5000$, $n = 1$M, $\delta = 0.1$, β of SMB(p^*) is 74.1% and that of SMB(0.5) is 54.3%, meaning that SMB(p^*) has 19.8% more probability to bound the cardinality estimate by a relative error of within 10% than SMB(0.5). This advantage of the generalized SMB over SMB(0.5) is also guaranteed under other values of n and m.

VI. Performance Evaluation

We evaluate the performance of the proposed SMB through experiments on a computer with Inter Core Xeon W-2135 3.7GHz and 32 GB memory. We also compare it with the state-of-the-art, i.e., MRB, FM, HLL++ and HLL-TailC under various performance metrics.

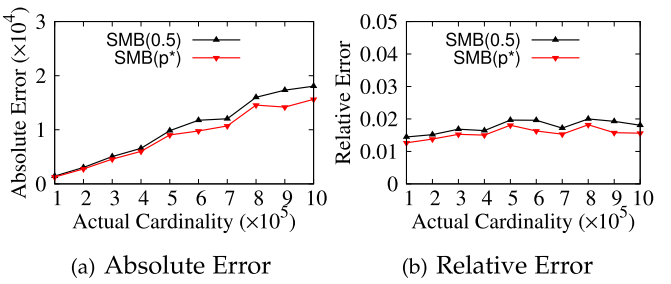
A. Experiment Setup

MRB, FM, HLL++, HLL-TailC and SMB all estimate the cardinality of a data stream. The data stream under practical scenarios can be of different types, such as a set of queries for a keyword, a set of tags in RFID system, etc, which we stress will not affect cardinality estimators' performance. In our experiments, the data stream contains randomly generated strings within the length of 128, each acting as a data item. The cardinality of the data stream, denoted as n, is the number of distinct strings in the data stream. n varies and its maximum value is 1M, which supports performance evaluation on data streams with very large cardinalities.

Parameter settings for FM, HLL++ and HLL-TailC follow the recommendation of [1], [31] and [20], [27] and can be found in Table I. The parameter setting under different n and m for MRB [3], [4] follows the recommendation of the original paper. The sampling probability of each bitmap in MRB [3], [4] is recommended as $1, \frac{1}{2}, \frac{1}{2^2} \ldots$ for high estimation accuracy. For SMB, the optimal values of T and p are given in Table II under different n and m. We evaluate the performance of MRB, FM, HLL++, HLL-TailC and SMB under different values of m and n. m can be 10000, 5000, 2500, and 1000. We employ four metrics: 1) Recording Throughput. The number of items recorded by the measurement module per second. The unit is data items per second (dps) or million data items per second (Mdps); 2) Query Throughput. For each arrival data item, we do a query operation to obtain the cardinality estimate. Query throughput represents the number of data items queried per second; 3) Estimation Error. It is categorized into two groups, absolute error and relative error. Let $\hat{n}$ be the estimated stream cardinality and n be the actual stream cardinality. The absolute error is defined as $|\hat{n} - n|$ and the relative error is defined as $\frac{|\hat{n}-n|}{n}$. The estimation error shows how the estimate deviates from the actual cardinality; 4) Estimation Bias. It is evaluated by the relative bias. The relative bias for a data stream is defined as $\frac{\hat{n}-n}{n}$. The estimation bias shows to what extent the estimate is underestimated/overestimated.

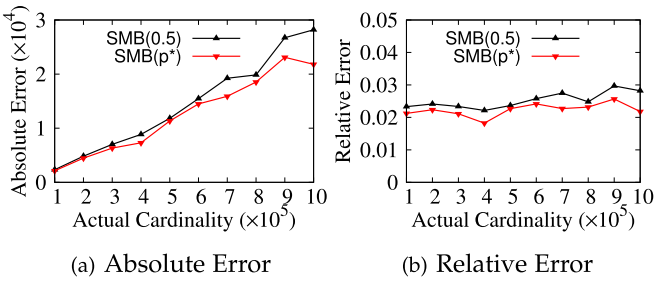
B. Comparison Between SMB(0.5) and SMB(p^*)

Denote SMB(p) as SMB that decreases sampling probability by p every time it proceeds to the next round. In particular,



(a) Absolute Error (b) Relative Error

Fig. 7. Estimation error comparison of SMB(0.5) and SMB(p^*) when allocated 10000 bits. By plot(a), SMB(p^*) reduces the absolute error by up to 18.5% compared to SMB(0.5).



(a) Absolute Error (b) Relative Error

Fig. 8. Estimation error comparison of SMB(0.5) and SMB(p^*) when allocated 5000 bits. By plot(a), SMB(p^*) reduces the absolute error by up to 22.5% compared to SMB(0.5).

TABLE III

Recording Throughput (Mdps) of SMB(p^*) and SMB(0.5) for Different Stream Cardinalities

Cardinality	SMB(p^*)	SMB(0.5)
10^2	18.1	19.2
10^3	15.3	26.7
10^4	42.3	44.5
10^5	58.6	60.9
10^6	69.5	73.3

TABLE IV

Query Throughput (Dps) of SMB(p^*) and SMB(0.5). The Results Will Not Be Affected by the Memory Size and Stream Cardinality

	SMB(p^*)	SMB(0.5)
Throughput	1.34×10^8	1.34×10^8

SMB(0.5) represents what Section III proposes. For the generalized SMB in Section IV, we derive the optimal setting of p, denoted as p^*, under given n and m, according to the analysis in Section V. The values of p^* are given in Table II. We compare SMB(0.5) and SMB(p^*) in terms of estimation accuracy under $n \in [10^5, 10^6]$ and $m = 10000$, 5000 bits, respectively. The results are shown in Figures 7 and 8. As we can see, the estimation error of SMB can be reduced by up to 18.5% for $m = 10000$ and 22.5% for $m = 5000$, by changing p from 0.5 to the optimal setting p^*. This demonstrates the practical values of the generalized design of SMB in Section IV. *In the remaining of this section, we will use SMB(p^*) as the representative of SMB to compare with*

TABLE V

RECORDING THROUGHPUT (MDPS) OF SMB FOR DIFFERENT STREAM CARDINALITIES IN COMPARISON WITH MRB, FM, HLL++ AND HLL-TAILC

Cardinality	MRB	FM	HLL++	HLL-TailC	SMB
10^2	19.2	19.3	8.11	7.92	18.1
10^3	19.1	19.2	8.01	7.99	15.3
10^4	19.3	19.7	8.26	8.11	42.3
10^5	20.2	20.2	8.38	7.96	58.6
10^6	20.9	20.1	8.24	8.10	69.5

TABLE VI

QUERY THROUGHPUTS (DPS) OF MRB, FM, HLL++, HLL-TAILC AND SMB UNDER DIFFERENT MEMORY ALLOCATION (BIT)

Memory	MRB	FM	HLL++	HLL-TailC	SMB
10000	3.95×10^6	1.01×10^6	0.91×10^4	0.78×10^4	1.34×10^8
5000	3.81×10^6	1.77×10^6	1.70×10^4	1.50×10^4	1.31×10^8
2500	3.71×10^6	4.33×10^6	4.42×10^4	3.89×10^4	1.30×10^8
1000	4.14×10^6	7.01×10^6	8.70×10^4	7.83×10^4	1.31×10^8

other cardinality estimation algorithms, i.e., FM, HLL++, MRB, HLL-TailC and without confusion, SMB is SMB(p^*). We also compare the recording and query throughputs of SMB(p^*) and SMB(0.5). The results in Tables III and IV show that both estimators have the similar throughput results. The reason is that, for the recording throughput, the generalized SMB only changes the geometric hash function to the generalized geometric hash function. But both hash functions are implemented by a uniform hash function, and thus they have similar computing overhead. For the query throughput, the estimation formulas for SMB(0.5) in (8) and for SMB(p^*) in (12) are very similar.

C. Recording Throughput

The recording throughput results under different stream cardinalities are listed in Table V. $m = 5000$ for all cardinality estimators. We stress that the recording throughput is not affected by m, as each cardinality estimator only operates on one unit (register for FM, HLL++ and HLL-TailC, bit for MRB and SMB). The results show that with the stream cardinality increasing, the recording throughput of SMB increases dramatically, while those of other cardinality estimators remain stable. The reason is that MRB, FM, HLL++ and HLL-TailC keep the same recording operation when the stream cardinality increases, while SMB can adaptively adjust the sampling probability. For data streams with large cardinalities, on average, SMB samples items with a small sampling probability. This explains why SMB can record more data items per second, especially when the stream cardinality is large. For instance, when the stream cardinality is 10^6, SMB increases the recording throughput by 232%, 245%, 743% and 758%, respectively, compared to MRB, FM, HLL++ and HLL-TailC.

D. Query Throughput

We evaluate the query throughputs of MRB, FM, HLL++, HLL-TailC and SMB. In our experiments, MRB maintains a counter array to keep the number of ones for each bitmap. Our additional maintenance of counters will not affect the accuracy but dramatically improve the query throughput of MRB.

TABLE VII

QUERY THROUGHPUT (DPS) OF SMB FOR DIFFERENT STREAM CARDINALITIES USING MEMORY OF 5000 BITS, IN COMPARISON WITH MRB, FM, HLL++ AND HLL-TAILC

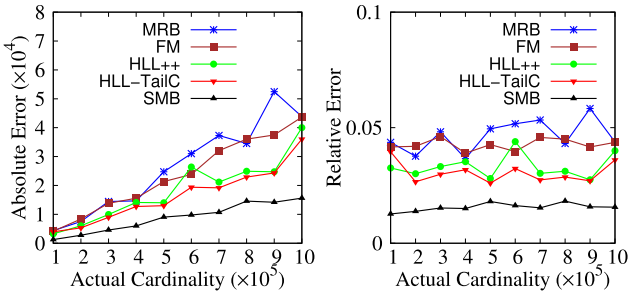
Card.	MRB	FM	HLL++	HLL-TailC	SMB
10^2	2.34×10^6	1.71×10^6	1.76×10^4	1.50×10^4	1.34×10^8
10^3	2.46×10^6	1.76×10^6	1.78×10^4	1.53×10^4	1.30×10^8
10^4	2.96×10^6	1.78×10^6	1.74×10^4	1.48×10^4	1.30×10^8
10^5	3.81×10^6	1.77×10^6	1.70×10^4	1.46×10^4	1.31×10^8
5×10^5	5.02×10^6	1.76×10^6	1.68×10^4	1.51×10^4	1.35×10^8
10^6	5.63×10^6	1.77×10^6	1.80×10^4	1.55×10^4	1.29×10^8

The query throughputs of five cardinality estimators when m is 10000, 5000, 2500, and 1000 are listed in Table VI. n is set as 10^5 and we will evaluate the impacts of n shortly. The results show that the query throughputs of FM, HLL++ and HLL-TailC are affected by memory allocation, while those of MRB and SMB are not. The reason is that FM, HLL++ and HLL-TailC need to collect the information in all registers for producing the cardinality estimate, while MRB needs to query an array of counters and SMB only needs to access two counters, i.e., r and v. This also explains why SMB stands out among all cardinality estimators in terms of query throughput. The results show that SMB' query throughput can be 130M per second, while HLL++ and HLL-TailC can only reach the query throughput by less than 0.1M per second. SMB improves the query throughput by at least 1500 times faster, making SMB suitable for online cardinality estimation. SMB also improves the query throughput a lot compared to MRB that has highest query throughput among existing work. The results show that The fast query supported by SMB allows instant identification of anomalies of cardinality and enables service administrator to respond to the anomaly in real-time.

We also investigate the impact of stream cardinality n on the query throughput which is presented in Table VII. Only MRB's query throughput is affected by the value of n. When n increases, the query throughput increases as well. The reason is that, MRB with k bitmaps needs to determine the most suitable sampling probability p_i with $0 \leq i < k$. When n is large, p_i is small, i.e,. $i \rightarrow k-1$. Consequently, MRB will query fewer counters. Although MRB's query throughput increases when n is large, it is still much smaller than that of SMB. MRB can only query less than 5% of items that SMB can query at the same time.

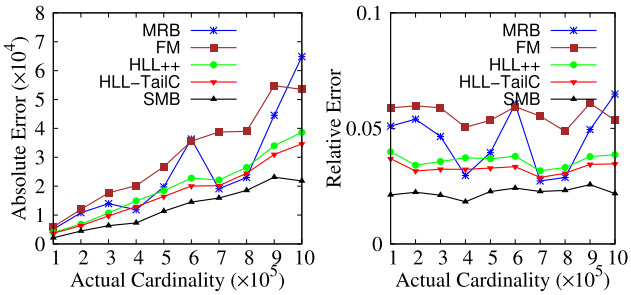
E. Estimation Error

The estimation error performances of all cardinality estimators when m is 10000 and 5000 are presented in Figures 9-10 where we plot the absolute error and relative error distribution with respect to the actual stream cardinality. Each point in the figure represents the average experimental result under 100 data streams with the same cardinality. The results show that SMB is the winner in terms of the estimation error, outper-



(a) Absolute Error (b) Relative Error

Fig. 9. Estimation error comparison of MRB, FM, HLL++, HLL-TailC and SMB when allocated 10000 bits. By plot(a), SMB reduces the absolute error by up to 74.0%, 73.1%, 61.1% and 52.9%, respectively, compared to MRB, FM, HLL++ and HLL-TailC.



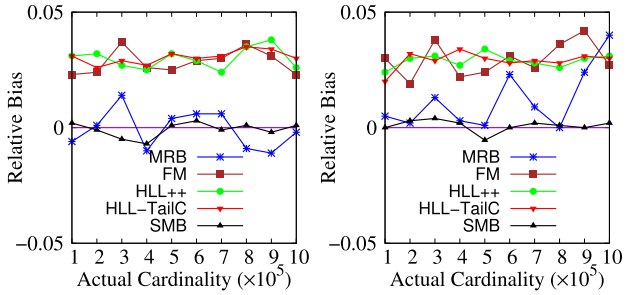
(a) Absolute Error (b) Relative Error

Fig. 10. Estimation error comparison of MRB, FM, HLL++, HLL-TailC and SMB when allocated 5000 bits. By plot(a), SMB reduces the absolute error by up to 70.9%, 71.6%, 45.6% and 42.6%, respectively, compared to MRB, FM, HLL++ and HLL-TailC.

forming the most accurate solutions, HLL++ and HLL-TailC. Besides, consider the estimation error of MRB. Its estimation errors for data streams with different cardinalities vary a lot, even if we use the average results among 100 data streams for each point. For instance, in Figure 9(a), when stream cardinality is 4×10^5, the absolute error of MRB is 54233. By comparison, when stream cardinality is 5×10^5, the absolute error of MRB is 10406. The result validate our argument that utilizing all the recorded information (SMB) promises a more accurate estimate compared to only using the information in the bitmap with the most sampling probability (MRB).

F. Estimation Bias

The relative bias performances of MRB, FM, HLL++, HLL-TailC and SMB when m is 10000 and 5000 are shown in Figure 11. We also plot function $y = 0$ to show the zero-bias line. Our findings are, SMB can produce the cardinality estimate with almost zero bias. The relative biases of data streams with different cardinalities produced by SMB are all within $[-0.01, 0.01]$. By comparison, FM and HLL++ produce positively biased stream cardinality estimates. For instance, the average relative biases of FM, HLL++ and HLL-TailC are around 0.03 under different memory allocations. MRB also produces biased cardinality estimates.



(a) 10000 bits (b) 5000 bits

Fig. 11. Relative bias of MRB, FM, HLL++, HLL-TailC and SMB w.r.t. actual cardinality under 10000-bit and 5000-bit memory allocation. SMB produces near non-biased cardinality estimation. FM's, HLL++'s and HLL-TailC's estimates are positively biased. The relative bias of MRB is up to 0.04 when allocated 5000 bits.

TABLE VIII

RECORDING THROUGHPUTS (MDPS) OF MRB, FM, HLL++, HLL-TAILC AND SMB UNDER THE CAIDA DATASET

Solutions	MRB	FM	HLL++	HLL-TailC	SMB
Throughput	24.3	22.3	6.34	6.01	29.9

TABLE IX

RECORDING THROUGHPUT (MDPS) OF SMB FOR DATA STREAMS OF THE CAIDA DATASET IN DIFFERENT CARDINALITY RANGES

Cardinality	>1	$> 10^3$	$> 5\times10^3$	$> 10^4$	$> 5\times10^4$
Throughput	29.9	34.2	45.5	48.7	65.3

TABLE X

QUERY THROUGHPUTS (DPS) OF MRB, FM, HLL++, HLL-TAILC AND SMB UNDER THE CAIDA DATASET

Solutions	MRB	FM	HLL++	HLL-TailC	SMB
Throughput	3.24×10^6	1.80×10^6	1.78×10^4	1.50×10^4	1.32×10^8

TABLE XI

AVERAGE ABSOLUTE ERRORS OF MRB, FM, HLL++, HLL-TAILC AND SMB FOR DATA STREAMS WITH CARDINALITIES ≤ 1000, UNDER DIFFERENT MEMORY ALLOCATION (BIT)

Memory	MRB	FM	HLL++	HLL-TailC	SMB
10000	0.039	0.10	0.03	0.02	0.01
5000	0.03	0.16	0.04	0.03	0.01
2500	0.05	0.24	0.07	0.06	0.03
1000	0.115	0.41	0.13	0.12	0.06

G. Results Under CAIDA Dataset

This subsection investigates their overall performance under a certain stream cardinality distribution, where each data stream is allocated with a cardinality estimator. We conduct experiments using real Internet traffic trace downloaded from CAIDA [40]. The traffic trace lasts for 10mins and contains around 200M packets. The packet in the traffic trace is the data item in the data stream model. We categorize the packets into different data streams by their destination addresses. That is, packets with the same destination address form a data stream. In each data stream, the packet is distinguished by the source

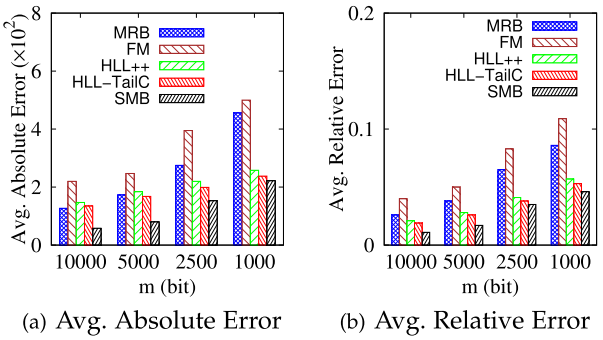


Fig. 12. Estimation error w.r.t. m in comparison with MRB, FM, HLL++, HLL-TailC and SMB.

address carried by the packet header. The stream cardinality is the number of distinct source addresses that contact the same destination address. By this categorization, the CAIDA dataset contains around 400k data streams and the largest cardinality among all data streams is around 80k.

Recording throughput: m is 5000 and the results are shown in Table VIII. SMB improves the recording throughput by 23.0%, 34.1%, 371.6% and 397.5%, respectively, compared to MRB, FM, HLL++ and HLl-TailC. We stress that SMB adaptively decreases the sampling probability during the recording of the data stream. When the stream cardinality is very large, its sampling probability becomes very small, which can dramatically increase the recording throughput. Therefore, the recording throughput of SMB is affected by the distribution of data streams in the dataset. Most data streams in the CAIDA dataset are with small cardinalities, which makes the recording throughput smaller. For more details, we present Table IX to show the recording throughput of SMB for data streams in different cardinality ranges. The results show when recording data streams with large cardinalities, the recording throughput increases dramatically.

Query throughput: The query throughputs of MRB, FM, HLL++, HLL-TailC and SMB are shown in Table X. m is 5000 for each cardinality estimator. As we can see, SMB improves the query throughput by 39.7 times, 72.3 times, 7314 times, and 8799 times, respectively, compared to MRB, FM, HLL++ and HLL-TailC.

Estimation error: We divide the data streams in the CAIDA dataset into two groups. One contains all data streams whose cardinalities are $\leq$1000. The other contains all data streams whose cardinalities are $>$1000. The reason is that, when the stream cardinality is small, FM, HLL++ and HLL-TailC are usually reduced to bitmap, and the sampling probabilities of MRB and SMB are 1 or close to 1. Therefore, their cardinality estimates are similar and very accurate. Specifically, FM reduces the 32-bit register to a bit. If all bits in the FM are zero, the register is reduced to a bit of zero; otherwise, the register is reduced to a bit of one. An FM with t FM registers is reduced to a bitmap with t bits. HLL++ and HLL-TailC follows the same reduction processing. m varies from 1000 to 10000, and the corresponding values of T for SMB follow the optimal setting in Table II. Table XI presents results for data streams whose cardinalities are $\leq$1000. The average absolute errors of all cardinality estimators are less than 1, regardless of the memory allocation.

The experimental results for data streams whose cardinalities are $>$1000 are shown in Figure 12. SMB is the most accurate cardinality estimator, regardless of the memory allocation. SMB reduces the average absolute error by up to 44.6%, 79.8%, 49.1% and 45.1%, respectively, compared to MRB, FM, HLL++ and HLL-TailC.

VII. Conclusion

This paper proposes a new design for online cardinality estimation in data streaming. It progressively decreases the sampling probability from 100% as data items are recorded, enabling sampling large-cardinality data streams with small probability and small-cardinality ones with large probability. We theoretically derive the estimation error bound and show that the estimation error is usually very small by illustrations. We also implement the proposed design and conduct experiments using two datasets. The experimental results show SMB is the best in all performance metrics and can achieve tremendous performance improvement (50% estimation error reduction, an order of magnitude higher throughput) in at least one metrics under the same memory allocation, compared to the best prior work.

References

[1] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, Oct. 1985.

[2] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[3] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2003, pp. 153–166.

[4] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[5] C. Qian, H. Ngan, Y. Liu, and L. M. Ni, "Cardinality estimation for large-scale RFID systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1441–1454, Sep. 2011.

[6] L. Wang *et al.*, "Fine-grained probability counting for cardinality estimation of data streams," *World Wide Web*, vol. 22, no. 5, pp. 2065–2081, Sep. 2019.

[7] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 101–114.

[8] J. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 561–575.

[9] C. Ma, H. Wang, O. O. Odegbile, and S. Chen, "Virtual filter for non-duplicate sampling," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–10.

[10] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 3, pp. 1–34, Dec. 2019.

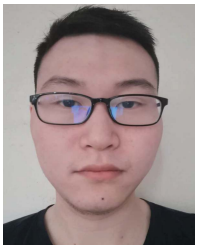
[11] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, Jun. 2015, pp. 417–428.

[12] H. Dai, M. Li, and A. Liu, "Finding persistent items in distributed datasets," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1403–1411.

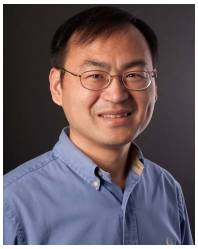
[13] H. Dai, M. Li, A. X. Liu, J. Zheng, and G. Chen, "Finding persistent items in distributed datasets," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 1–14, Feb. 2020.

[14] C. Ma, H. Wang, O. Odegbile, and S. Chen, "Noise measurement and removal for data streaming algorithms with network applications," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9.
[15] C. Ma, S. Chen, Y. Zhang, Q. Xiao, and O. O. Odegbile, "Super spreader identification using geometric-min filter," *IEEE/ACM Trans. Netw.*, early access, Aug. 31, 2021, doi: 10.1109/TNET.2021.3108033.
[16] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," *ACM Trans. Sensor Netw.*, vol. 4, no. 2, pp. 1–40, Mar. 2008.
[17] N. Ntarmos, P. Triantafillou, and G. Weikum, "Counting at large: Efficient cardinality estimation in internet-scale data networks," in *Proc. 22nd Int. Conf. Data Eng. (ICDE)*, Apr. 2006, p. 40.
[18] S. Ganguly, M. Garofalakis, R. Rastogi, and K. Sabnani, "Streaming algorithms for robust, real-time detection of DDoS attacks," in *Proc. 27th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2007, p. 4.
[19] N. Zhao *et al.*, "Automatically and adaptively identifying severe alerts for online service systems," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 2420–2429.
[20] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. 16th Int. Conf. Extending Database Technol.*, 2013, pp. 683–692.
[21] P. Rob, S. Dorward, R. Griesemer, and S. Quinlan, "Interpreting the data: Parallel analysis with Sawzall," *Sci. Program.*, vol. 13, no. 4, pp. 277–298, 2005.
[22] S. Melnik *et al.*, "Dremel: Interactive analysis of web-scale datasets," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 330–339, Sep. 2010.
[23] A. Hall, O. Bachmann, R. Büssow, S. Gănceanu, and M. Nunkesser, "Processing a trillion cells per mouse click," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1436–1446, Jul. 2012.
[24] Z. Bar-Yossef, T. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Proc. Int. Workshop Randomization Approximation Techn. Comput. Sci.* Berlin, Germany: Springer, 2002, pp. 1–10.
[25] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2007, pp. 199–210.
[26] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proc. Eur. Symp. Algorithms*. Berlin, Germany: Springer, 2003, pp. 605–617.
[27] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. Int. Conf. Anal. Algorithms*, Nancy, France, 2007, pp. 137–156.
[28] H. Harmouch and F. Naumann, "Cardinality estimation: An experimental survey," *Proc. VLDB Endowment*, vol. 11, no. 4, pp. 499–512, Dec. 2017.
[29] A. Metwally, D. Agrawal, and A. E. Abbadi, "Why go logarithmic if we can go linear? Towards effective distinct counting of search traffic," in *Proc. 11th Int. Conf. Extending Database Technol., Adv. Database Technol.*, 2008, pp. 618–629.
[30] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *Proc. 29th ACM SIGMOD-SIGACT-SIGART Symp. Princ. Database Syst. Data (PODS)*, 2010, pp. 41–52.
[31] Q. Xiao, Y. Zhou, and S. Chen, "Better with fewer bits: Improving the performance of cardinality estimation of large data streams," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.
[32] Q. Xiao, S. Chen, Y. Zhou, and J. Luo, "Estimating cardinality for arbitrarily large data stream with improved memory efficiency," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 433–446, Apr. 2020.
[33] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," Presented at the 10th USENIX Symp. Networked Syst. Design Implement. (NSDI), 2013.
[34] L. Tang, Q. Huang, and P. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Toronto, ON, Canada, 2020, pp. 1608–1617.
[35] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proc. VLDB Endowment*, vol. 14, no. 6, pp. 1040–1052, Feb. 2021.
[36] R. B. Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu, "Routing oblivious measurement analytics," in *Proc. IFIP Netw. Conf. (Networking)*, Jun. 2020, pp. 449–457.
[37] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1253–1264, Oct. 2011.
[38] M. Goemans. (2021). *Harmonic Number, Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Harmonic_number
[39] S. Janson, "Tail bounds for sums of geometric and exponential variables," *Statist. Probab. Lett.*, vol. 135, pp. 1–6, Apr. 2018.
[40] UCSD. (2015). *Caida UCSD Anonymized 2015 Internet Traces on Jan. 17*. [Online]. Available: https://www.caida.org/data/passive/passive_2015_dataset.xml

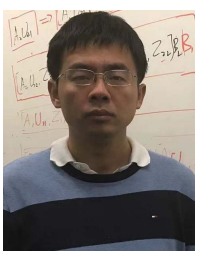
Haibo Wang (Graduate Student Member, IEEE) received the B.E. degree in nuclear science and the master's degree in computer science from the University of Science and Technology of China, China, in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Science and Engineering, University of Florida. His main research interests include internet traffic measurement, software-defined networking, and optical circuit scheduling.



Chaoyi Ma (Graduate Student Member, IEEE) received the B.S. degree in computer information security from the University of Science and Technology of China in 2018. He is currently pursuing the Ph.D. degree in computer and information science and engineering with the University of Florida. His advisor is Prof. Shigang Chen. His research interests include big data, network traffic measurement, computer network security, and data privacy in machine learning.



Shigang Chen (Fellow, IEEE) received the B.S. degree in computer science from the University of Science and Technology of China in 1993 and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, Champaign, IL, USA, in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining the University of Florida in 2002. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. He holds 13 U.S. patents, and many of them were used in software products. He has published over 200 peer-reviewed journals/conference papers. His research interests include big data, the Internet of Things, cybersecurity, RFID technologies, and intelligent cyber-transportation systems. He received the NSF CAREER Award and several best paper awards. He served as an Associate Editor for IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, and a number of other journals. He served in various chair positions or a committee members for numerous conferences. He is an ACM Distinguished Scientist.



Yuanda Wang received the master's degree in computer science from NYU in 2015. He is currently pursuing the Ph.D. degree with the Department of Computer and Information Science and Engineering, University of Florida. His main research interests include edge computing and internet traffic measurement.