Online Cardinality Estimation by Self-morphing Bitmaps

Haibo Wang  Chaoyi Ma  Shigang Chen  Yuanda Wang
Department of Computer & Information Science & Engineering,
University of Florida, Gainesville, FL 32611, USA,
Email: {wanghaibo, ch.ma}@ufl.edu  sgchen@cise.ufl.edu  yuandawang@ufl.edu

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, popularity tracking on social media, and cache optimization in proxy servers. 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 evaluate the self-morphing bitmap theoretically and experimentally. The results demonstrate that it significantly outperforms the prior art.

I. INTRODUCTION

Cardinality estimation is one of the fundamental problems in the data streaming field [1], [2], [3], [4], [5], [6], [7], [8]. 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 [9], each keyword in the sequence of searches received by a web search engine [10], each file (name) accessed by users at a data storage center [11], [12], each tag ID collected from an RFID system [5], [13], etc.

Cardinality estimation has many practical applications. For instance, consider Internet traffic received by a router. 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 this 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 the stream surges, it may signal a DDoS attack as an abnormal number of distinct sources access the service hosted at the destination address. In yet another example, if a search engine treats all search records on the same keyword as a data stream and the client IP address in each record as a data item, the stream’s cardinality suggests the popularity of the keyword being searched.

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 [14], such as Sawzall [15], Dremel [16], and PowerDrill [17], estimate the cardinalities of very large data sets on a daily basis. As pointed out in the paper [14], 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, 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 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 \[13, 19\]. 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 [20], HyperLogLog (HLL) [21] and HyperLogLog++ (HLL++) [14], 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 \[22\]. 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. Compared to HLL++ \[14\] 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, e.g., $D = \{d_1, d_2, d_1, d_1, \ldots\}$, 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. Let $D$ be an example. When $d_1$ and $d_2$ first appear, the stream cardinality is respectively incremented by one. When the third data item, i.e., $d_1$, arrives, since it is the second appearance of $d_1$, we should be able to find that out and ignore it. The problem is to design a data structure called cardinality estimator that records the data items of a stream and estimates the stream cardinality.

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.

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 $B[H(d)] = 1$ to 1. When being queried, bitmap traverses all the bits and calculates the number of ones in the array, denoted as $U$. The cardinality estimate $\hat{n}$ by bitmap \[2\] is

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

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 \[23\].
Multi-resolution Bitmap (MRB). MRB [3, 4] employs \( k \geq 1 \) bitmaps, denoted as \( B_0, B_1, \ldots, B_{k-1} \), each with a distinct sampling probability \( p_i, \forall 0 \leq i < k \), to simultaneously record a data stream and selects one of them for producing the cardinality estimate. Given the total memory allocation \( m \) bits, each \( i \)-th bitmap \( B_i \) is \( \frac{m}{2^i} \) (suppose it is integer) bits long. Considering the convenience of implementation and estimation accuracy, MRB recommends that \( p_1 = \frac{1}{2}, \forall 0 \leq i < k \) [3, 4] and \( p_0 > p_1 > \ldots > p_{k-1} \). For any data item \( d \), it gets sampled by \( B_i \) if \( p_i > \frac{H(d)}{W} \mod \), 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 \( B_j[H(d)] = 1, \forall 0 \leq j \leq i \) with \((i+1)\) updates, MRB only sets \( B_i[H(d)] = 1 \) with single update. Consequently, \( B_0, B_1, \ldots, B_{i-1} \) 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. 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 \( U_i \geq T \), where \( T \) is a predefined threshold. If no \( B_i \) is found, \( i = 0 \). \( B_i \) is considered as the bitmap with most sampled probability. MRB collects the number of sampled distinct data items under the sampling probability \( p_i \), which are recorded in \( B_i \), \( B_{i+1}, \ldots, B_{k-1} \), \( \forall i \leq j \leq k-1 \), each with a cardinality estimate of \( -\frac{m}{k} \ln(1 - \frac{U_j}{m/k}) \). 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 \( U_{k-1} = \frac{m}{2^k} \), which is \( 2^{k-1} \frac{m}{2^k} \ln(2^k) \). 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 2 if \( G(x) \) is an integer and \( G(x) = i, i \geq 0 \), 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 32 bits, with the \( i \)-th bit denoted as \( F[i], 0 \leq i \leq 31 \). For any item \( d \), we calculate \( G(d) \) with \( G(d) \leq 31 \) and set \( F[G(d)] = 1 \). Consider the cardinality estimate of FM. Roughly speaking, any item is hashed to the \( i \)-th bit with probability \( \frac{1}{2^i} \) 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, \( t = \frac{m}{32} \) (suppose it is an integer). For query, FM produces the cardinality estimate as
\[
\hat{n} = t \cdot \frac{z_{t+1}^{-1} - 1}{\phi} \tag{3}
\]
where \( z_t \) is the number of consecutive ones of \( F_i \) starting from the least significant bit, and \( \phi \) is pre-computed constant that is related to \( t \), \( \phi = 0.78 \) when \( t \) is large enough. Refer to [11] for the value of \( \phi \) under different \( t \).

An HLL++ register \( Y \) is a counter of 5 bits (for estimation range up to \( 2^{256} - 1 \) or \( c > 5 \) bits if the stream cardinality is up to \( 2^{256} - 1 \)), which represents an integer of range \( 0 \leq Y \leq 31 \). It is a compact version of the FM register. For any data item \( d \), we calculate \( G(d) \) with \( G(d) \leq 30 \) 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++, \( t = \frac{m}{32} \) (suppose it is an integer). For query, HLL++ produces cardinality estimate through arithmetic mean of \( Y \) of \( Y_0, \ldots, Y_{t-1} \) as
\[
\hat{n} = \alpha_t \cdot t \cdot \bar{Y} \quad \text{with} \quad \bar{Y} = t \left( \sum_{i=0}^{t-1} 2^{-Y_i} \right)^{-1} \tag{4}
\]
where \( \alpha_t \) is a constant that can be calculated as \( \alpha_t = 0.7213 / (1 + \frac{0.677}{t}) \) when \( t \geq 128 \). For value of \( \alpha_t \) under different \( t \), refer to [21, 14]. 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. 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 \( Y'_i = Y_i - \bar{B} \), where variable \( \bar{B} \) is maintained and is equal to the \( \min_{0 \leq i < t} Y_i \). If \( Y' \) overflows, \( Y'_i = 15 \). For query, HLL-TailC recovers \( Y_i \) based on \( Y'_i \) and \( B \) 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 \( \frac{1}{4}, \frac{1}{8}, \frac{1}{32}, \frac{1}{256}, \frac{1}{32}, \frac{1}{4}, \frac{1}{4}, \ldots \), respectively. However, unlike HLL++ which has fixed coefficient \( \alpha_t \), Refined HLL needs to first use a portion of the data stream to learn 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
To reduce the variance, MinCount divides the hash values of the LogLog family and bits, under the same accuracy requirement. This from an practical perspective is not successful as the actual memory consumption is not reduced. Moreover, the effort from a practical perspective is not successful as the current method has 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.

### 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 a number of data streams [27, 9, 28, 29, 30]. 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. 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-ins for these sketches and the performance improvement by our work can benefit these sketches accordingly.

---

**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 $2^{12}$.**

Note that $H$ represents the overhead for one hash operation and $A$ 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.

---

**Summary of existing cardinality estimators and our goal.** Table I compares the recording and query overhead of each cardinality estimator where the number of hash operations $H$ and the memory (bit) accessed $A$ per data item are adopted as the metrics as hash and memory access are two major computation-hungry operations. 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 V). But its estimation accuracy is not as high as HLL++. Moreover, an experimental survey [22] 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 alone 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.
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

Sampling rate

\[ p_{k,i} = \frac{1}{2^{k-i}} \]

\[ B_{k,i} \]

Array of \( k \) bitmaps

\[ p_i = \frac{1}{2} \]

\[ B_i \]

\[ p_i = \frac{1}{4} \]

\[ B_i \]

\[ p_i = \frac{1}{2} \]

\[ B_i \]

\[ p_0 = 1 \]

\[ B_0 \]

\[ m/k \]

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., \( B_3, ..., B_{k-1} \) are wasted.

Sampling rate

\[ p_{k,i} = \frac{1}{2^{k-i}} \]

\[ B_{k,i} \]

Array of \( k \) bitmaps

\[ p_i = \frac{1}{2} \]

\[ B_i \]

\[ p_i = \frac{1}{2^{k-i}} \]

\[ B_{i-1} \]

\[ p_i = \frac{1}{2} \]

\[ B_i \]

\[ p_0 = 1 \]

\[ B_0 \]

\[ m/k \]

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

It is well known that the Bitmap estimator \([2],[31]\) has a limited estimation range of \(-m \ln m\) \([2]\), which is a serious problem in practice for large data streams \([9]\), 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

\[ \begin{align*}
  p_{k,i} &= \frac{1}{2^{k-i}} \\
  p_i &= \frac{1}{2} \\
  p_i &= \frac{1}{4} \\
  p_i &= \frac{1}{2} \\
  p_0 &= 1
\end{align*} \]

The memory space occupied by other bitmaps, i.e., \( B_{i+1}, ..., B_{k-1} \) 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 \([2],[9]\). 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 \( p_1 = \frac{4}{3} \) 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 a logical bitmap $L$ 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. ∀0 ≤ $i < k$, if we find the number of bits set to ones in $T$ logical bitmap $L_i$ reaches the threshold $T$, we will reduce the sampling probability from $p_i = \left(\frac{1}{2}\right)^i$ to $p_{i+1} = \left(\frac{1}{2}\right)^{i+1}$. 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 $p_i = \left(\frac{1}{2}\right)^i$, 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.

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 [5] 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 $i$th round will have a sampling probability $p_r = \frac{1}{r}$. 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 Section I), 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 $\frac{1}{2}$, $\frac{1}{4}$, ... .

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.

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.

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 $d_0, d_2, d_3, d_4, d_5$ 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 \( G(d_8) = 0 < r \)).

Algorithm 1 Recording a data item in SMB

1: Input: data item \( d, T \)
2: Action: record \( d \), and update \( r \) and \( v \)
3: if \( G(d) \geq r \) then
4: \( L_0[H(d)] = 0 \) then
5: \( L_0[H(d)] = 1 \)
6: \( v = v + 1 \)
7: \( r = r + 1 \)
8: \( v = 0 \)
9: end if
10: end if
11: end if

Algorithm 2 Querying on the SMB

1: Input: \( T, S \)
2: Output: the cardinality estimate of the data stream
3: return \( S[r] - 2' m \ln(1 - \frac{v}{m - r'}) \)

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 \( p_i = \frac{1}{r}, 0 \leq i \leq r \).

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

The proof can be easily derived and we defer it to Section VII-A 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

\[
U_i = \left\{ \begin{array}{ll}
T, & 0 \leq i < r \\
v, & i = r
\end{array} \right.
\]

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

\[
\hat{n}_i = -m_i \ln(1 - U_i/m_i)
\]

From (5), the estimate in the \( i \)th round of recording is constant if \( i < r \), which can be computed in advance.

\[
\hat{n}_i = -m_i \ln(1 - T/m_i), \quad 0 \leq i < r
\]

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 \( m_i/m \). Therefore, the cumulative estimate in the first \( r \) rounds of recording is

\[
\sum_{i=0}^{r-1} \frac{m_i}{p_i m_i} = \sum_{i=0}^{r-1} -2'm \ln(1 - \frac{T}{m_i})
\]

We maintain an array \( S \) to keep the above constants under different \( r \), where

\[
S[r] = \begin{cases} 
\sum_{i=0}^{r-1} -2'm \ln(1 - \frac{T}{m_i}), & r > 0 \\
0, & r = 0
\end{cases}
\]

Through \( S \), we only need to calculate the estimate in the \( r \)th round of recording, i.e., \( \hat{n}_r \).

\[
\hat{n}_r = -m_r \ln(1 - \frac{v}{m_r}) = -(m - rT) \ln(1 - \frac{v}{m - rT})
\]

Aggregating the estimate in each round, we obtain the cardinality estimate \( \hat{n} \).

\[
\hat{n} = \sum_{i=0}^{r} \frac{m_i}{p_i m_i} \hat{n}_i = S[r] - 2'm \ln(1 - \frac{v}{m - rT})
\]

The query operation is described in Algorithm 2. Comparing with the existing work HLL++, and HLL-TaiC with query overhead of \( mA \), 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 32A. Comparing with MRB which only uses the information in the bitmap with the best sampling probability, SMB utilizes all the recorded information (see [1]), 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.

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 \( r = \frac{m}{k} - 1 = k - 1 \) (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 (\( 2^{k-1} \frac{m}{k} \ln(\frac{m}{k}) \)).

**Theorem 2:** For any data item \( d_i \), 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. Obviously, \( G(d') = G(d'') = G(d) \). 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.
In the original paper, for a fair comparison, the optimal setting for parameter estimation performance of SMB and interprets the theorem. After that, we present the optimal setting for parameter $T$.

### A. Estimation Error Bound

**Theorem 3**: 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 $\beta$, i.e.,

$$
\Pr\left(\left|\frac{n - \hat{n}}{n}\right| \leq \delta \right) \geq \beta = 1 - 2e^{-\frac{(1 - \delta)(1 + \delta)\ln(2\pi\delta\delta^{-1})}{2}},
$$

where $r$ is the maximum integer value that satisfies $n(1 + \delta) \geq S[r]$ and $U_r \leq T$ is the maximum integer value that satisfies $n(1 + \delta) \geq S[r] + 2(m/T - |n - m/T|)$ (but not larger than $T$) and the value of array $S$ can be found in [9].

The proof is deferred to Section VII-B. Next, we interpret Theorem 3. According to (12), the error bound is closely related to variable $m$, $T$ and $n$. For ease of understanding the error bound, we also give visual display by plotting $\beta$ with respect to $\delta$ when $n=1M$. We choose four representative values of $m$, i.e., $m=10k$, $5k$, $2.5k$ and $1k$ with the unit of bit and $T$ is optimally set. The setting of $T$ 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 $\delta$ with high probability. For instance, when $m=10000$ bits and $\delta=0.1$, $\beta=0.971$. That means $|\frac{n - \hat{n}}{n}| < 0.1$ happens with the probability of $\geq 97.1\%$. Even when $m$ is small, i.e., $m=1000$, $|\frac{n - \hat{n}}{n}| < 0.30$ happens with the probability of $\geq 80.2\%$. 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. 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++ [21], [14], the standard error $|\frac{n - \hat{n}}{n}|$ is given, from which we can bound the relative error by $\delta$ with a probability $\beta$ using Chebyshev’s inequality. We plot $\beta$ with respect to $\delta$ for SMB, MRB and HLL++ when $n = 1M$ and $m = 10000$ for each algorithm in Figure 5(b), which shows under the same $\delta$, SMB’s $\beta$ is larger than that of MRB and HLL++. That means SMB is more likely to bound the estimation error with an arbitrary constant $\delta$ than MRB and HLL++.

### B. Parameter Setting for $T$

This subsection discusses how $T$ is set. The SMB can support maximum $\frac{T}{\delta}$ rounds of recording, which should be larger than or equal to $r+1$, i.e., $\frac{T}{\delta} \geq r+1$. 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 $\beta$ maximized. We use numerical computing to derive the optimal setting of $m/T$ under different values of $m$ and $n$, which are listed in Table II. The setting under specific values of $m$ and $n$ can also be obtained in the same manner. 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 (12), we know $\beta$ is affected by $\left(\frac{m/m_r}{m_r - m/m_r}\right)\frac{n_m}{2m}$, where $\frac{2m}{m_r - m/m_r}$ represents the expected number of distinct data items required make $v$ increases from $U_r - 1$ to $U_r$. The ratio is $O(\frac{1}{n_r})$ and always stays very large, meaning that the beta is guaranteed under when $n$ varies.

### V. 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], [24] and [21], [14] 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, which is given in Table III. The sampling probability of each bitmap in MRB [3], [4] is recommended as $\frac{1}{n}$, $\frac{1}{2n}$, $\frac{1}{4n}$ ... for high estimation accuracy. For SMB, the optimal value of $T$ is given in Table I 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. $n$ can be up to 1M. 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 $|n - \hat{n}|$ and the relative error is defined as $\frac{|n - \hat{n}|}{n}$. 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 $\frac{\hat{n}}{n} - 1$. The estimation bias shows to what extent the estimate is underestimated/overestimated.

### Table III: Parameter setting of MRB: number of bitmaps $m/k$ and length for each bitmap $m/k$ under given $n, m$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m/k$</th>
<th>$k$</th>
<th>$m/k$</th>
<th>$k$</th>
<th>$m/k$</th>
<th>$k$</th>
<th>$m/k$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1M</td>
<td>909</td>
<td>11</td>
<td>416</td>
<td>178</td>
<td>14</td>
<td>66</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>900k</td>
<td>909</td>
<td>11</td>
<td>416</td>
<td>192</td>
<td>13</td>
<td>66</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>800k</td>
<td>909</td>
<td>11</td>
<td>416</td>
<td>192</td>
<td>13</td>
<td>66</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>700k</td>
<td>909</td>
<td>11</td>
<td>416</td>
<td>192</td>
<td>13</td>
<td>71</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>600k</td>
<td>1000</td>
<td>10</td>
<td>416</td>
<td>192</td>
<td>13</td>
<td>71</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>500k</td>
<td>1000</td>
<td>10</td>
<td>454</td>
<td>208</td>
<td>12</td>
<td>71</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>400k</td>
<td>1000</td>
<td>10</td>
<td>454</td>
<td>208</td>
<td>12</td>
<td>71</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>300k</td>
<td>1111</td>
<td>9</td>
<td>500</td>
<td>208</td>
<td>12</td>
<td>76</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>200k</td>
<td>1111</td>
<td>9</td>
<td>500</td>
<td>227</td>
<td>11</td>
<td>83</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>100k</td>
<td>1428</td>
<td>7</td>
<td>555</td>
<td>250</td>
<td>10</td>
<td>90</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>80k</td>
<td>1428</td>
<td>7</td>
<td>625</td>
<td>277</td>
<td>9</td>
<td>90</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

### Table IV: Recording throughput (Mdps) of SMB for different stream cardinalities in comparison with MRB, FM, HLL++, and HLL-TailC.

<table>
<thead>
<tr>
<th>Cardinality</th>
<th>MRB</th>
<th>FM</th>
<th>HLL++</th>
<th>HLL-TailC</th>
<th>SMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^4</td>
<td>19.2</td>
<td>19.3</td>
<td>8.11</td>
<td>7.92</td>
<td>19.2</td>
</tr>
<tr>
<td>10^5</td>
<td>19.1</td>
<td>19.2</td>
<td>8.01</td>
<td>7.99</td>
<td>26.7</td>
</tr>
<tr>
<td>10^6</td>
<td>19.3</td>
<td>19.7</td>
<td>8.26</td>
<td>8.11</td>
<td>44.5</td>
</tr>
<tr>
<td>10^7</td>
<td>20.2</td>
<td>20.2</td>
<td>8.38</td>
<td>7.96</td>
<td>60.9</td>
</tr>
<tr>
<td>10^8</td>
<td>20.9</td>
<td>20.1</td>
<td>8.24</td>
<td>8.10</td>
<td>73.3</td>
</tr>
</tbody>
</table>

### Table V: Query throughputs (dps) of MRB, FM, HLL++, HLL-TailC and SMB under different Memory allocation (bit).

<table>
<thead>
<tr>
<th>Card.</th>
<th>MRB</th>
<th>FM</th>
<th>HLL++</th>
<th>HLL-TailC</th>
<th>SMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>10^4</td>
<td>3.95×10^9</td>
<td>1.01×10^10</td>
<td>0.91×10^10</td>
<td>0.78×10^10</td>
<td>1.34×10^10</td>
</tr>
<tr>
<td>10^5</td>
<td>3.81×10^10</td>
<td>1.77×10^10</td>
<td>1.70×10^10</td>
<td>1.50×10^10</td>
<td>1.31×10^10</td>
</tr>
<tr>
<td>10^6</td>
<td>3.71×10^10</td>
<td>4.33×10^10</td>
<td>4.42×10^10</td>
<td>3.89×10^10</td>
<td>1.30×10^10</td>
</tr>
<tr>
<td>10^7</td>
<td>4.14×10^10</td>
<td>7.01×10^10</td>
<td>8.70×10^10</td>
<td>7.83×10^10</td>
<td>1.31×10^10</td>
</tr>
</tbody>
</table>

### Table VI: Query throughput (dps) of SMB for different stream cardinalities using memory of 5000 bits, in comparison with MRB, FM, HLL++ and HLL-TailC.

B. Recording Throughput

The recording throughput results under different stream cardinalities are listed in Table IV $n=5000$ 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 10^9, SMB increases the recording throughput by 250%, 364%, 789% and 804%, respectively, compared to MRB, FM, HLL++, and HLL-TailC.

C. 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.

The query throughputs of five cardinality estimators when $m$ is 10000, 5000, 2500, and 1000 are listed in Table V $n$ is set as 10^9 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 0.1 M 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 throughput which is presented in Table VI. Only 0.1% 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.

D. Estimation Error

The estimation error performances of all cardinality estimators when $m$ is 10000 and 5000 are presented in Figures 6-7 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, outperforming 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 6(a), when stream cardinality is $4 \times 10^5$, the absolute error of MRB is 54233. By comparison, when stream cardinality is $5 \times 10^5$, 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).

E. Estimation Bias

The relative bias performances of MRB, FM, HLL++, HLL-TailC and SMB when $m$ is 10000 and 5000 are shown in Figure 8. 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.

F. 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 [32]. The traffic trace lasts for 10 mins and contains around 200 M 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 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 30.0%, 39.4%, 390.5% and 417.5%, 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 VIII 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 IX. \( m \) is 5000 for each cardinality estimator. As we can see, SMB improves the query throughput by 42.6%, 76.8%, 46.1% and 41.1%, respectively, compared to MRB, FM, HLL++ and HLL-TailC. By plot(a), SMB reduces the average absolute error by up to 42.6%, 76.8%, 46.1% and 41.1%, 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 \( T \) follow the optimal setting in Table IX. Table X 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 9. SMB is the most accurate cardinality estimator, regardless of the memory allocation. SMB reduces the average absolute error by up to 42.6%, 76.8%, 46.1% and 41.1%, respectively, compared to MRB, FM, HLL++ and HLL-TailC.

<table>
<thead>
<tr>
<th>Solutions</th>
<th>MRB</th>
<th>FM</th>
<th>HLL++</th>
<th>HLL-TailC</th>
<th>SMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>3.24×10^11</td>
<td>1.80×10^10</td>
<td>1.78×10^10</td>
<td>1.50×10^10</td>
<td>1.32×10^10</td>
</tr>
</tbody>
</table>

**TABLE VIII**: Recording throughput (Mbps) of SMB for data streams of the CAIDA dataset in different cardinality ranges.

**TABLE IX**: Query throughputs (bps) of MRB, FM, HLL++, HLL-TailC and SMB under the CAIDA dataset.

**TABLE X**: Average absolute errors of MRB, FM, HLL++, HLL-TailC and SMB for data streams with cardinalities \( \leq 1000 \), under different memory allocation (bit).

---

**VI. 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.

**VII. PROOF**

**A. Proof of Lemma 7**

Consider the \( i \)th round of recording. Only the data item \( d \) whose geometric hash value \( G(d) \geq i \) can pass Step 1, which happens with probability of \( \Pr(G(d) \geq i) \). For \( i = 0 \), we have \( \Pr(G(d) \geq 0) = 1 \); for \( i > 0 \), we have \( \Pr(G(d) \geq i) = 1 - (1/2 + \ldots + 1/2^i) = 1/2^i \)
B. Proof of Theorem 3

There are \( r+1 \) rounds of recording in the bitmap. Let \( \hat{n} \) be the estimated stream cardinality. From \([5]\), we have

\[
\hat{n} = \sum_{i=0}^{r} 2^i \frac{m}{m_i} \hat{m}_i \sum_{i=0}^{r} 2^i \frac{m}{m_i} (m_i \ln(m_i/(m_i-U_i)))
\]

Let \( X^j_i \) 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 \( r \)th 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, \( X^j_i \) is a geometric random variable and we have

\[
E(X^j_i) = \frac{m}{(m_i-j+1)} p_i = \frac{m^2^j}{(m_i-j+1)}.
\]

Totally, there are \( rT \) variables from the first \( r \) rounds of recording, i.e., \( X^0_1, X^1_2, ..., X^r_i, X^1_{r-1}, ..., X^r_{r-1}, X^{r-1}_r, ..., X^r_{U_r} \), and \( U_r \) variables in the \( r \)th round of recording, i.e., \( X^1_r, X^2_r, ..., X^{U_r}_r \). These variables are mutually independent. Denote

\[
X = \sum_{0 \leq i < r, 1 \leq j \leq T} X^j_i + \sum_{1 \leq j \leq U_r} X^j_r.
\]

When the recording process of the data item terminates, the event \( X^i_{U_r} \) happens and the event \( X^i_{U_r+1} \) does not happen. Therefore, the actual stream cardinality locates between \( X \) and \( X + X^U_r \). 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

\[
X = n.
\]

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 \([8]\), we have

\[
\Pr(X \geq (1 + \delta) E(X)) \leq e^{-\frac{(mx - U_r + 1)}{2m} E(X)(\delta - \ln(1 + \delta))}
\]

\[
\Pr(X \leq (1 - \delta) E(X)) \leq e^{-\frac{(mx - U_r + 1)}{2m} E(X)(-\delta - \ln(1 - \delta))},
\]

where \( \delta \in (0,1) \) and variable \( p_* \) in paper \([3]\) represents the minimum success probability of all geometric variables, which is \( (\frac{m - U_r + 1}{2m}) \) in the context of this proof. Usually, we bound \( X \) by a small \( \delta \) and hence we have \( \ln(1 + \delta) = \delta + \frac{\delta^2}{2} + o(\delta^2) \) and \( \ln(1 - \delta) = -\delta + \frac{\delta^2}{2} + o(\delta^2) \). Therefore, the above equations can be combined together as

\[
\Pr(|X - E(X)| \geq \delta E(X)) \leq 2e^{-\frac{(mx - U_r + 1)}{2m} E(X) \frac{\delta^2}{2}}. \tag{16}
\]

The following lemma proves \( E(X) = \hat{n} \) as long as the \( m \) is sufficiently large.

**Lemma 4:** \( \lim_{m \rightarrow \infty} E(X) = \hat{n} \).

**Proof:** Consider the \( i \)th round of recording with \( 0 \leq i < r \). From \([14]\), we know

\[
E(\sum_{1 \leq j \leq T} X^j_i) = \sum_{1 \leq j \leq T} \frac{2^i m}{m_i - j + 1} = \frac{2^i m}{\sum_{1 \leq j \leq T} m_i - j + 1}
\]

\[
= 2^i m (H_{m_i} - H_{m_i - r}) \tag{17}
\]

Note that \( H_x \) is the \( x \)-th harmonic number. Since the number of bits in \( L_i \) is usually sufficiently large, e.g., \( 10^4 \), using the asymptotics of the harmonic numbers \([34]\), we obtain

\[
E(\sum_{1 \leq j \leq T} X^j_i) = 2^i m (H_{m_i} - H_{m_i - r})
\]

Similarly, for the \( r \)th round of recording, we have \( E(\sum_{1 \leq j \leq U_r} X^j_r) = 2^r \frac{m}{m_r} \hat{n} \). 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 \( \lim_{m \rightarrow \infty} E(X) = \hat{n} \).

From Lemma 4 and \([15], \[16]\) can be rewritten as

\[
\Pr\left(\frac{|n - \hat{n}|}{n} \geq \delta \right) \leq 2e^{-\frac{(mx - U_r + 1)}{2m} \frac{\delta^2}{2}}
\]

\[
\Pr\left(\frac{|n - \hat{n}|}{n} \leq \frac{\delta}{2} \right) \leq 2e^{-\frac{(mx - U_r + 1)}{2m} \frac{\delta^2}{2}}
\]

\[
\Pr\left(\frac{|n - \hat{n}|}{n} \leq \frac{\delta}{2} \right) \leq 1 - 2e^{-\frac{(mx - U_r + 1)}{2m} \frac{\delta^2}{2}} \tag{18}
\]

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 \( \hat{n} \approx n(1 + \delta) \). We have

\[
n(1 + \delta) = \hat{n} \geq S[r] \]

where \( r \) is upper-bound by the maximum integer value that makes the above inequality hold.

Consider the maximum value of \( U_r \) with \( U_r \leq T \). Since \( \hat{n} \approx (1 + \delta)n \), we can have the upper bound for \( U_r \).

\[
n(1 + \delta) \geq \hat{n} \geq S[r] + 2^r m (-\ln \frac{mx - U_r}{m_r}),
\]