# *Taking One Small Step Forward:* Finding Low-Frequency Items in Data Streams

*Abstract*—**With the explosive growth of the amount of data and the rapid development of information technology, low-frequency items in data streams have attracted much attention due to their large amount and the rich information contained in them. However, as far as we know, there is no method finding low-frequency items in data streams. To address this problem, we propose an one-pass algorithm, called *BFSS*, which can find the items in a data stream whose frequency is no more than a user-specified support approximately. *BFSS* is simple and has small memory footprints. Although the output is approximate, we can guarantee no false negatives (*FNs*) and only a few false positives (*FPs*) exist in the result. Given a little modification, *BFSS* can be extended to *SBFSS*, which can handle data streams in limited and small space and guarantee only a few *FPs* and theoretically bounded number of *FNs* exist in the result. Experimental results on real-world dataset show *BFSS* achieves high performance using much less space compared to *nCount* and *rCount*, which are two naive methods finding low-frequency items in data streams. *SBFSS* outperforms *lCount*, a method finding low-frequency items in data streams using limited space with a simple replacement strategy, in both precision and recall.**

## I. Introduction

In many real-world applications, information such as web click data [1], stock ticker data [2], [3], sensor network data [4], phone call records [5], and network packet traces [6] appears in the form of data streams. Motivated by the above applications, researchers started working on novel algorithms for analyzing data streams. Problems studied in this context include approximate frequency moments [7], distinct values estimation [8], [9], bit counting [10], duplicate detection [11], [12], approximate quantiles [13], wavelet based aggregate queries [14], correlated aggregate queries [15], frequent elements [16]–[19] and top-$k$ queries [20], [21]. However, to the best of our knowledge, there is no algorithm finding low-frequency items in data streams. In fact, low-frequency items in data streams have attracted much attention because of the rich information they contain which can be seen through the formulas of entropy of data streams [22]. We take one small step forward to find low-frequency items in data streams approximately in this paper.

### A. Motivating Examples

*1) Individualized demands mining:* With the rapid development of internet, we can shop and search whatever we are interested in online. Our demands, for example, buying a regular water glass online or searching the information about a tourist attraction etc, are popular most of the time and these demands can be easily satisfied. However, we are no longer satisfied with just popular demands nowadays. For example, it is not so easy for us to buy embroidery stitches or find the information about a nameless village in China online, because they are individualized demands.

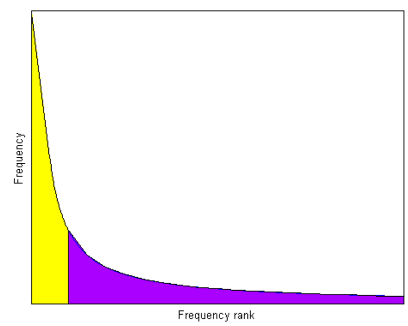



Fig. 1: Rank-frequency distribution

Microsoft lags so far behind Google in the search engine market because it focused a lot on the head of the queries and didn't acknowledge the long tail, said Yusuf Mehdi, senior vice president of the Online Audience Group for Microsoft Bing, at Search Engine Strategies[1]. For Microsoft, focusing on the head instead of the "long tail" meant that it returned popular queries but failed to satisfy less common queries. The long tail of queries, or the individualized demands, ended up yielding more sizable traffic and therefore more money for Google over the last 11-plus years.

From above, we can see that individualized demands, in some sense, are more important than popular demands, and finding individualized demands requires the algorithms finding low-frequency items in data streams.

*2) Distribution estimation:* Distribution is a basic property of a data stream, therefore it is important to estimate the distribution of a data stream efficiently. Sampling is a simple and fast method to estimate distributions, however, it may cause significant errors sometimes. Fig.1 is a power-law graph, being used to demonstrate the ranking of frequencies of items in data streams under power-law distributions[2]. From Fig.1, we observe that low-frequency items have a great influence on distributions, and combining the information of both frequent and low-frequency items, we can have a relatively more comprehensive estimation of the distribution of a data stream.

In fact, *BFSS* can approximately find both frequent and low-frequency items in data streams, given a little modification, *BFSS* can be extended to estimate the percentage of the total frequencies of a certain proportion of items in data streams, for example, we can approximately estimate the

[1]http://www.eweek.com/c/a/Search-Engines/Microsoft-Ignored-the-Long-Tail-in-Search-Bing-Boss-Says-396023

[2]the distributions of a wide variety of physical, biological, and man-made phenomena are approximately power-law distributions

percentage of the total frequencies of the most frequent twenty[3] percent of items in a data stream, which is similar to the expression of the famous Pareto principle, also known as the 80-20 rule.

### B. Our Contributions

We are the first to pose and formally define the problem *s-Bounded Low-Frequency Elements (s-BLFE)*, the formal definition of which will be given later, and to the best of our knowledge, there is no method solving the problem till now.

We propose *BFSS*, which extends the classic algorithm *Space Saving* to maintain both frequent and low-frequency items in a data stream approximately. The basic idea of our solution is as follows: each item in a data stream is either a frequent one or a low-frequency one once the threshold $s[\in (0,1)]$ is confirmed, so we can maintain low-frequency items by filtering the frequent items out. A major problem we have to deal with is to maintain an itemset, items in which appear in the data stream, and this can be done approximately using a Bloom filter. *BFSS* gurantees no false negatives and provably few false positives using small memory footprints.

However, the size of a Bloom filter must increase with the alphabet's size in order to keep low false positive rate, and here comes a problem: In many embedded devices, such as sensors and routers etc., the storage space is limited and small, in which case *BFSS* will produce many *FPs*. Inspired by the method presented in [11], we propose *SBFSS* which extends *BFSS* to deal with data streams in limited and small space, and *SBFSS* gurantees a few false positives and theoretically bounded number of false negatives.

### C. Roadmap

In Section 2, we present problem statement and some backgrounds on the existing approches which deal with the problem *$\epsilon$-Deficient Frequent Elements*. Our solutions are presented and discussed in Section 3. In Section 4, we experimentally evaluate our methods. Conclusions are given in Section 5.

## II. PRELIMINARIES

This section presents problem statement and some representative algorithms solving *$\epsilon$-Deficient Frequent Elements* [23] which will be formally defined below. Table I summarizes the major notations in this paper.

### A. Problem Statement

Consider an input stream $S = e_1, e_2, ..., e_N$ of current length $N$, which arrives item by item. Let each item $e_i$ belong to a universe set $A = \{a_1, a_2, ..., a_M\}$ of size $M$.

The problem *s-BLFE* can be stated as follows: given a data stream $S$ along with two user-specified parameters: a support parameter $s[\in (0,1)]$ and an error parameter $\epsilon[\in (0,1)]$ such that $\epsilon \le s$.

At any point of time, with a small bounded memory, output a list of items with the following guarantees:

[3]"twenty" is just an example, and the specific value varies and will be explained later

TABLE I: Major Notations Used in the Paper.

| Notation | Meaning |
|---|---|
| *BFSS* | Our first algorithm |
| *SBFSS* | Our second algorithm |
| $S$ | The input data stream |
| $A$ | The alphabet of $S$ |
| $M$ | The size of $A$ |
| $n$ | The number of distinct items in $S$ |
| $s$ | The user-specified support parameter |
| $\epsilon$ | The user-specified error parameter |
| $D$ | The synopsis used in *BFSS* and *SBFSS* |
| $e$ | The item monitored in $D$ |
| $f(e)$ | The estimated frequency of $e$ |
| $\Delta(e)$ | The estimated error of $f(e)$ |
| $E$ | The item set monitored in $D$ |
| $min$ | The minimum value of $f(e)$ in $D$ |
| $C$ | The maximum number of counters in $D$ |
| $m$ | The number of counters used in $D$ |
| $N$ | The length of $S$ |
| $H$ | The number of hash functions used in *BFSS* and *SBFSS* |
| $f_S(e)$ | The Frequency of item $e$ in $S$ |
| *FPs* | The items in $S$ with frequency more than $\lfloor sN \rfloor$ wrongly output |
| *FNs* | The items in $S$ with frequency no more than $\lfloor sN \rfloor$ wrongly neglected |
| *BF* | The Bloom filter used in *BFSS* |
| $K$ | The size of *BF* |
| $p_1$ | The percentage of the number of the items whose frequency is above $\lfloor sN \rfloor$ |
| $p_2$ | The percentage of the total frequencies of the items whose frequency is above $\lfloor sN \rfloor$ |
| *SBF* | The Stable Bloom filter used in *SBFSS* |
| $K'$ | The size of *SBF* |
| $k$ | Number of cells used in *SBF* |
| $d$ | Number of bits allocated per cell |
| $max$ | The value a cell is set to |
| $P$ | Number of cells we pick to decrement by 1 in each iteration |
| $p$ | The probability that a cell is picked to be decremented by 1 in each iteration |
| $p'$ | The probability that a cell is set in each iteration |
| $W$ | The real data set we used in our experiments |

1. all items whose true frequency is no more than $\lfloor sN \rfloor$ are output.
2. no item whose true frequency is no less than $\lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ is output.

Imagine a user who is interested in identifying all items whose frequency is no more than $0.1\%N$. Then $s = 0.1\%$. The user is free to set $\epsilon$ to whatever she feels is a comfortable margin of error. As a rule of thumb, she could set $\epsilon$ to one-tenth or one-twentieth the value of $s$ and use our algorithm. Let us assume she chooses $\epsilon = 0.01\%$ (one-tenth of $s$). As per Property 1, all items with frequency no more than $0.1\%N$ will be output, there will be no false negatives. As per Property 2, no element with frequency no less than $\lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ will be output.

### B. Related Work

As far as we know there is no related algorithm addressing *s-BLFE*, however, the methods we propose are based on the algorithm solving *$\epsilon$-Deficient Frequent Elements* which can be described as follows: given an input stream $S$ of current length $N$ and a support threshold $s \in (0,1)$, return the items whose frequency is guaranteed to be no smaller than $\lfloor (s-\epsilon)N \rfloor$ deterministically or with a probability of at least $1-\delta$, where $\epsilon \in (0,1)$ is a user-defined error and $\delta \in (0,1)$ is a probability of failure, so we examine several algorithms solving *$\epsilon$-Deficient Frequent Elements*.

Research can be divided into two groups: *counter-based* techniques and *sketch-based* techniques.

**Counter-Based Techniques** keep an individual counter for each item in the monitored set, a subset of A. The counter of a monitored item, $e_i$, is updated when $e_i$ occurs in the stream. If there is no counter kept for the observed ID, it is either disregarded, or some algorithm-dependent action is taken.

Two representative algorithms *Sticky Sampling* and *Lossy Counting* were proposed in [23]. The algorithms cut the stream into rounds, and they prune some potential low-requency items at the edge of each round. Though simple and intuitive, they suffer from zeroing too many counters at rounds boundaries, and thus, they free space before it is really needed. In addition, answering a frequent elements query entails scanning all counters.

The algorithm *Space Saving* , the one we are based at, was proposed in [20]. The algorithm maintains a synopsis which keeps all counters in an order according to the value of each counter's monitoring frequency plus maximum possible error. For a non-monitored item, the counter with the smallest counts, *min*, is assigned to monitor it, with the items monitoring frequency $f(e)$ set to 1 and its maximal possible error $\Delta(e)$ set to *min*. Since $min \leq \epsilon N$ (this follows because of the choice of the number of counters), the operation amounts to replacing an old, potentially infrequent item with a new, hopefully frequent item. This strategy keeps the item information until the very end when space is absolutely needed, and it leads to the high accuracy of *Space-Saving*. Experiments done in [24], [25] showed *Space-Saving* outperformed other Counter-Based techniques in recall and precision tests.

**Sketch-Based Techniques** do not monitor a subset of items, rather provide, with less stringent guarantees, frequency estimation for all items using bitmaps of counters. Usually, each item is hashed into the space of counters using a family of hash functions, and the hashed-to counters are updated for every hit of this item. Those representative counters are then queried for the item frequency with less accuracy, due to hashing collisions.

The *Count-Min Sketch* algorithm of Cormode and Muthukrishnan [26] maintains an array of $d \times w$ counters, and pairwise independent hash functions $h_j$ map items onto $[w]$ for each row. Each update is mapped onto $d$ entries in the array, each of which is incremented. The Markov inequality is used to show that the estimate for each $j$ overestimates by less than $n/w$, and repeating $d$ times reduces the probability of error exponentially.

The *hCount* algorithm was proposed in [27]. The data structure and algorithms used in *Count-Min Sketch* and *hCount* shared the similarity, but were simultaneously and independently investigated with different focuses.

## III. OUR ALGORITHMS

In this section, we will discuss our approaches *BFSS* and *SBFSS* in detail.

### A. Challenges of $s$-BLFE

*$s$-BLFE* has two main challenges due to the different features between frequent items and low-frequency items over data streams.

**The Long Tails in data streams**. It can be easily proved that there are at most $\lceil 1/s \rceil$ frequent items whose frequency is more than $\lfloor sN \rfloor$ in any data stream, however, there is no upper bound of the number of the low-frequency items whose frequency is no more than $\lfloor sN \rfloor$. In fact, our experiments show that the low-frequency items occupy most of the distinct items in data streams, and it is almost impossible to maintain all of them in memory. Another observation is their frequencies are very low and close as well, and it may consume much space to separate low-frequency items from frequent items especially when $s$ is very small.

**Unpredictability**. A basic and common idea of *Counter-Based Techniques* is to discard potential infrequent items dynamically, and it is based on the fact that potential infrequent items will never become frequent items if they don't appear afterwards, however, this fact no longer applies to low-frequency items in data streams because frequent items will possibly become low-frequency items if they don't appear afterwards. The unpredictability of low-frequency items makes it difficult to maintain them directly.

### B. The BFSS Algorithm

In consideration of the challenges in *$s$-BLFE*, we tried to solve the problem indirectly by filtering frequent items out which is the underlying idea of *BFSS*.

Two algorithms are proposed for updating and outputting results separately. Algorithm 1 maintains a Bloom filter $BF$ of size $K$ with $H$ uniformly independent hash functions $\{h_1(x), ..., h_H(x)\}$ and a synopsis $D$ with $C$ counters. Each of these $H$ hash function maps an item from $A$ to $[0, ..., K-1]$. Initially each bit of BF is set to 0 and $D$ has $C$ empty counters. Each newly arrived item in the stream is mapped to $H$ bits in BF by the $H$ hash functions and we set the $H$ bits to 1. Then if we observe an item that is monitored in $D$, we just increment $f(e)$. If we observe an item, $e_{new}$, that is not monitored in $D$, handle it depending on whether there is an empty counter in $D$. If there is one, we just allocate it to $e_{new}$ and set $f(e_{new})$ to 1 and set $\Delta(e_{new})$ to 0. Otherwise, we just replace the item that currently has the least hits, $min$, with $e_{new}$. Assign $f(e_{new})$ the value $min + 1$ and assign $\Delta(e_{new})$ the value $min$.

Algorithm 2 checks and outputs the items whose frequency is no more than $sN$. For each item in $A$, we first check whether it is in $BF$. If the item is not in $BF$, it must not be a low-frequency item. If the item is in $BF$ but not in $D$, we output it as a low-frequency item. If the item appears both in $BF$ and $D$, we identify it as a low-frequency item if $f(e) \leq \lfloor sN \rfloor + \Delta(e)$ with high probability. The time requirement of Algorithm 2 is linear to the range of universe. It is acceptable when the frequency of the requests is not high.

### C. Analysis of BFSS

In this section, we present a theoretical analysis of *BFSS* described in Section III-B. We analyze *FNs*, *FPs*, space complexity, and time complexity. At last, we will identify the challenges to *BFSS*.

**Algorithm 1** BFSS Update Algorithm

**Input:** Stream $S$, support threshold $s$

$N = 0, m = 0, C = \lceil 1/\epsilon \rceil, K = \lambda', H = \mu'$; {$N$: the length of the input stream; $m$: the number of counters used in $D$; $C$: the maximum number of counters in $D$; $K$: size of Bloom filter; $H$: the number of hash functions; The value of $\lambda'$ and $\mu'$ will be discussed in detail later.}
The form of the counters in $D$ is $(e, f(e), \Delta(e))$
**for** $i = 0$ to $K - 1$ **do**
    $BF[i] = 0$
**end for**
**for** each item $e$ of stream $S$ **do**
    **for** $i = 1$ to $H$ **do**
        $BF[h_i(e)] = 1$
    **end for**
    **if** $e$ is monitored in $D$ **then**
        $f(e) = f(e) + 1$;
    **else if** $m < C$ **then**
        Assign a new counter $(e, 1, 0)$ to it
        $m = m + 1$
    **else**
        Let $e_m$ be the item with least hits, $min$
        Replace $e_m$ with $e$ in $D$
        $f(e) = min + 1, \Delta(e) = min$
    **end if**
    $N = N + 1$;
**end for**

**Algorithm 2** BFSS Query Algorithm

**Input:** *BF*, $D$, $s$, $A$, $N$, $M$
**Output:** low-frequency items with threshold $s$

$flag = true$; { *flag*: indicate whether an item is in *BF*}
**for** $i = 0$ to $M - 1$ **do**
    $flag = true$
    **for** $j = 1$ to $H$ **do**
        **if** $BF[h_j(A[i])] == 0$ **then**
            $flag = false$
            break;
        **end if**
    **end for**
    **if** $flag == true$ **then**
        **if** $A[i]$ is monitored in $D$ **then**
            **if** $f(A[i]) \leq \lfloor sN \rfloor + \Delta(A[i])$ **then**
                output $A[i]$ as a low-frequency item
            **end if**
        **else**
            output $A[i]$ as a low-frequency item
        **end if**
    **end if**
**end for**

*1) **Analysis of FNs:*** In this section, we will prove that there are no *FNs* in the output of *BFSS*. The proof is based on Lemma 1 to Lemma 5, and the detailed proof of Lemma 1 to Lemma 3 can be found in [20].

*Lemma 1:* $N = \sum_{\forall i | e_i \in E}(f(e_i))$

*Proof:* Every hit in $S$ increments only one counter by 1 among the $M$ counters which can be easily proved when $D$ has empty counters. It is true even when a replacement happens, i.e., the observed item $e$ was not monitored and $D$ has no empty counters, and it replaces another item $e_m$. This is because we add $f(e_m)$ to $f(e)$ and increment $f(e)$ by 1. Therefore, at any time, the sum of all counters is equal to the length of the stream observed so far. ■

*Lemma 2:* Among all counters in $D$, the minimum counter value, $min$, is no greater than $\lfloor \frac{N}{m} \rfloor$.

*Proof:* Lemma 1 can be written as:

$$min = \frac{N - \sum_{\forall i | e_i \in E}(f(e_i) - min)}{m} \tag{1}$$

All the items in the summation of Equation 1 are non negative because all counters are no smaller than $min$, hence $min \leq \lfloor \frac{N}{m} \rfloor$. ■

*Lemma 3:* For any item $e \in E$, $0 \leq \Delta(e) \leq \lfloor \epsilon N \rfloor$.

*Proof:* From Algorithm 1, $\Delta(e)$ is non-negative because any observed item is always given the benefit of doubt. $\Delta(e)$ is always assigned the value of the minimum counter at the time $e$ started being observed. Since the value of the minimum counter monotonically increases over time until it reaches the current $min$, then for all monitored items $\Delta(e) \leq min$.

Consider thses two cases: i) if $m = \lceil \frac{1}{\epsilon} \rceil$, $min \leq \lfloor \epsilon N \rfloor$; ii) if $m < \lceil \frac{1}{\epsilon} \rceil$, $\Delta(e) = 0$. For any case, we have $0 \leq \Delta(e) \leq \lfloor \epsilon N \rfloor$. ■

*Lemma 4:* For any item $e \notin E$ but appearing in $S$, $f_S(e) \leq min \leq \lfloor \epsilon N \rfloor$.

*Proof:* From Lemma 2, we can easily get $min \leq \lfloor \epsilon N \rfloor$ because there must be no empty counter in $D$ and $m = \lceil \frac{1}{\epsilon} \rceil$. We only have to porve that any item satisfying $f_S(e) > min$ must be monitored in $D$, i.e. $e \in E$. The proof is by contradiction. Assume $e \notin E$. Then it was evicted previously, and we assume that it had been monitored in $i (> 0)$ time slots, and $e$ appeared $n_j (0 < j \leq i)$ times in the $j$th time slot, therefore $n_j$ satisfies:

$$\sum_{j=1}^{i} n_j = f_S(e) \tag{2}$$

We assume that $\Delta(e_j) (\geq 0)$ denotes the error estimation assigned to $e$ at the start of the $j$th time slot, and we have the following inequality because the minimum counter value increases monotonically:

$$\begin{gathered} \Delta(e_1) + n_1 \leq \Delta(e_2) \\ \Delta(e_2) + n_2 \leq \Delta(e_3) \\ \ldots \\ \Delta(e_{i-1}) + n_{i-1} \leq \Delta(e_i) \\ \Delta(e_i) + n_i \leq min \end{gathered} \tag{3}$$

After adding up the left and right sides of inequality group 3, we can get:

$$\Delta(e_1) + \sum_{j=1}^{i} n_j \leq min \tag{4}$$

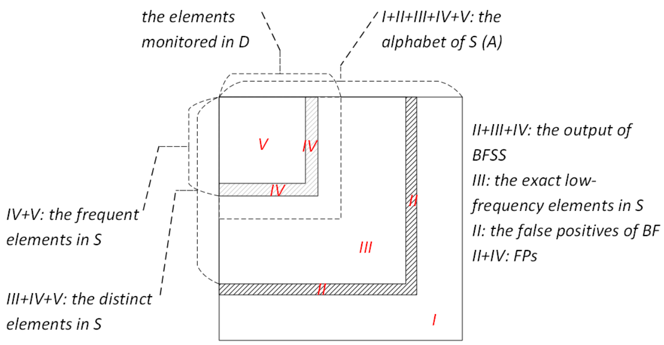



Fig. 2: The schematic diagram of the analysis of *BFSS*

Therefore $f_S(e) \leq min$, which contradicts the condition $f_S(e) > min$. ■

*Lemma 5:* For any item $e \in E$, $f(e) - \Delta(e) \leq f_S(e) \leq f(e) \leq f(e) - \Delta(e) + min$

*Proof:* Since the value of the minimum counter monotonically increases, we have $\Delta(e) \leq min$, which indicates $f(e) \leq f(e) - \Delta(e) + min$. $f(e) - \Delta(e)$ is the true frequency of $e$ since it was lastly observed, so $f(e) - \Delta(e) \leq f_S(e)$. From Lemma 4, we can find that $\Delta(e)$ over estimated the frequency of $e$ before it was observed, and it clearly indicates $f_S(e) \leq f(e)$. ■

*Theorem 1:* There are no *FNs* in the output of *BFSS*.

*Proof:* We only hava to prove that the items we don't output contain no low-frequency items. Algorithm 2 shows that two kinds of items are not ouput: i) the items filtered out by *BF*. ii) the items monitored in $D$ with $f(e) - \Delta(e) > \lfloor sN \rfloor$. The first kind of items are obviously not low-frequency items because they never appeared in $S$. From Lemma 5, we know that the items monitored in $D$ with $f(e) - \Delta(e) > \lfloor sN \rfloor$ must satisfy $f_S(e) > \lfloor sN \rfloor$, which means the second kind of items must be frequent items. ■

*2) **Analysis of FPs**:* In this section, we will give a theoretically bound of the expectation of the number of *FPs* in the output of *BFSS* regardless of the distribution of $S$, and a tighter bound can be derived for data streams under Pareto distribution [28].

*Lemma 6:* Any item $e$ with $f_S(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must not be output.

*Proof:* From Lemma 4, we know that any item $e$ with $f_S(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must be monitored in $D$, i.e. $e \in E$. Then from Lemma 5, we can get that the items monitored in $D$ must satisfy $f_S(e) \leq f(e)$, and that is to say any item $e$ monitored in $D$ with $f_S(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must satisfy $f(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$. At last from Lemma 3, we can prove that any item $e$ with $f(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must satisfy $f(e) > \lfloor sN \rfloor + \Delta(e)$, which means $e$ will not be output. ■

*Lemma 7:* The probability of a false positive of *BF* is no more than:

$$(1 - (1 - \frac{1}{K})^{HM})^H \tag{5}$$

*Proof:* A false positive of *BF* means an item in $A$ not appearing in $S$ but not filtered out by *BF*. Observe that after inserting $M$ keys into a table of size $K$, the probability that a particular bit is still 0 is exactly

$$(1 - \frac{1}{K})^{HM} \tag{6}$$

Hence the probability of a false positive in this situation is $(1 - (1 - \frac{1}{K})^{HM})^H$. However, we know that $M$ denotes the size of $A$ which is the alphabet of $S$, so the number of the distinct items in $S$ must be no more than $M$, i.e. $n \leq M$, and further we have $(1 - (1 - \frac{1}{K})^{Hn})^H \leq (1 - (1 - \frac{1}{K})^{HM})^H$. ■

*Theorem 2:* Assuming no specific data distribution, the expectation of the number of *FPs* in the output of *BFSS*, denoted as $E(\#FPs)$, satisfies:

$$E(\#FPs) < M(1 - (1 - \frac{1}{K})^{HM})^H + \lfloor \frac{1}{s} \rfloor \tag{7}$$

*Proof:* From Algorithm 1, we can observe that two kinds of items contribute to *FPs*: i) the false positives of *BF*; ii) the items with $f_S(e) > \lfloor sN \rfloor$ wrongly output. The two cases correspond to the gray areas in Fig. 2 which is abstracted out from the analysis of *BFSS*. Concretely speaking, the ligth gray area represents the items with $f_S(e) > \lfloor sN \rfloor$ wrongly output, i.e. the second case, and the dark gray area represents the items not filtered out by *BF*, i.e. the first case. Furthermore, the output of *BFSS* is represented by $II + III + IV$, and the exact low-frequency items in $S$ is represented by $III$.

For the first case, we define the independent 0-1 random variables $x_i (1 \leq i \leq M)$ for each item in $A$, and the value of $x_i$ depends on whether $a_i$ appeared in $S$ or not. If $a_i$ appeared in $S$, then $x_i = 0$; If not, then with a probability of $(1 - (1 - \frac{1}{K})^{Hn})^H$, $x_i = 1$; From the defination of $x_i$, we can find that the expectation of the number of the false positives of *BF* equals $E(\sum_{i=1}^{M} x_i)$, i.e. $(M - n)(1 - (1 - \frac{1}{K})^{Hn})^H$.

For the second case, we know from Lemma 6 that items with $f_S(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must not be not output, so only items with $\lfloor sN \rfloor < f_S(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ are likely to be output, and the maximum number of these items is $\lfloor \frac{1}{s} \rfloor$ because there are at most $\lfloor \frac{1}{s} \rfloor$ items with $f_S(e) > \lfloor sN \rfloor$ in $S$. From above, due to the linear properties of expectation, we can get:

$$E(\#FPs) \leq (M - n)(1 - (1 - \frac{1}{K})^{Hn})^H + \lfloor \frac{1}{s} \rfloor \tag{8}$$

In addition, $n \leq M$, so inequation 7 can be easily derived from inequation 8. ■

However, from above, we can see that $\frac{1}{s}$ is a very loose bound of the number of the items with $\lfloor sN \rfloor < f_S(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$. We will get a tighter bound if $S$ .

*Theorem 3:* Assuming noiseless Pareto data with parameter $\alpha$ and $f_S(e_m)$, where $\alpha (> 0)$ and $e_m$ denotes the item with the lowest frequency, we have:

$$E(\#FPs) < M(1 - (1 - \frac{1}{K})^{HM})^H + T \tag{9}$$

$$T = min\{((\frac{f_S(e_m)}{\lfloor sN \rfloor})^\alpha - (\frac{f_S(e_m)}{\lfloor sN \rfloor + \lfloor \epsilon N \rfloor})^\alpha)M, \lfloor \frac{1}{s} \rfloor\} \tag{10}$$

*Proof:* The Pareto distribution[4] has the following property: If $X$ is a random variable with a Pareto distribution, then the probability that $X$ is greater than some number $x$, i.e. the tail function, is given by:

$$Pr(X > x) = \begin{cases} (\frac{x_m}{x})^\alpha & x \geq x_m \\ 1 & x < x_m \end{cases}$$

where $x_m$ is the minimum possible value of $X$, and $\alpha$ is a positive parameter. In such case, the expected value of the number of the items with $\lfloor sN \rfloor < f_S(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ is $((\frac{f_S(e_m)}{\lfloor sN \rfloor})^\alpha - (\frac{f_S(e_m)}{\lfloor sN \rfloor + \lfloor \epsilon N \rfloor})^\alpha)n$. ■

From Inequation 7, we observe that the upper bound of $E(\#FPs)$ is related with the values of $H$ and $K$, in addtion, the value of $K$ directly influence the space consumption of *BFSS*, and the value of $H$ directly influence the update time of *BFSS*, so it is of great significance to choose the appropriate values of $K$ and $H$ to keep a good balance of the precision, the space consumption and the update time of *BFSS* .

*3) **Space complexity**:* In this section, we will analyze the space complexity of *BFSS* including how to choose the appropriate values of $H$ and $K$.

**Choosing the appropriate values of $H$ and $K$.** Our goal is to choose the appropriate values of $H$ and $K$ to keep $\#FPs$ as small as possible, meanwhile, we have to take the space consumption and update time into consideration as well. However, we can only get the upper bound of the value of $E(\#FPs)$, which is not equivalent to $\#FPs$. In fact, from the Chernoff bound, we know that there is a high probability that an independent random variable hovers near its expected value, which means we can approximately treat $E(\#FPs)$ as $\#FPs$.

Obviously, the upper bound of $E(\#FPs)$ given in Inequation 7 is not so tight, and the upper bound of $E(\#FPs)$ given in Inequation 8 is much tighter. However, it is difficult for us to choose the appropriate values of $H$ and $K$ to minimize the upper bound given in Inequation 8 because the value of $n$ is variable, and the task will be much easier for the bound given in Inequation 7, besides, the values of $M$ and $s$ are confirmed at the very beginning, therefore our task is to minimize the value of $(1 - (1 - \frac{1}{K})^{HM})^H$, denoted as $F$, and we have:

$$F \approx (1 - e^{-\frac{HM}{K}})^H = e^{H \ln(1 - e^{-\frac{HM}{K}})} \tag{11}$$

Let $P = e^{-\frac{HM}{K}}$ and $G = H \ln(1 - e^{-\frac{HM}{K}})$, and in order to minimize $F$, we only have to minimize $G$:

$$G = (-\frac{K}{M}) \ln(P) \ln(1 - P) \tag{12}$$

The first derivative of $G$ with respect to $P$, denoted as $G'$, is:

$$G' = \frac{K}{M}(\frac{1}{1-P} \ln(P) - \frac{1}{P} \ln(1 - P)) \tag{13}$$

From Equation 13, we can easily observe that when $P = \frac{1}{2}$, $G$ reaches its minimum value. In this case, we have:

$$H = \frac{K}{M} \times \ln 2 \tag{14}$$

[4] https://en.wikipedia.org/wiki/Pareto_distribution

TABLE II: The value of $F$ under various $\frac{K}{M}$ and $H$ combinations.

| $K/M$ | $H$ | $H = 8$ | $H = 9$ | $H = 10$ | $H = 11$ | $H = 12$ |
|---|---|---|---|---|---|---|
| 13 | 9.01 | 0.00199 | 0.00194 | 0.00198 | 0.0021 | 0.0023 |
| 14 | 9.7 | 0.00129 | 0.00121 | 0.0012 | 0.00124 | 0.00132 |
| 15 | 10.4 | 0.000852 | 0.000775 | 0.000744 | 0.000747 | 0.000778 |
| 16 | 11.1 | 0.000574 | 0.000505 | 0.00047 | 0.000459 | 0.000466 |
| 17 | 11.8 | 0.000394 | 0.000335 | 0.000302 | 0.000287 | 0.000284 |

Combined with Equation 11 and Equation 14, we have:

$$F \approx (1 - e^{-\frac{HM}{K}})^H = (\frac{1}{2})^H \approx (0.6185)^{\frac{K}{M}} \tag{15}$$

From the above derivation, we know that once the value of $\frac{K}{M}$ is confirmed, we can get the appropriate value of $H$ to minimize the value of $F$. For example, if the value of $\frac{K}{M}$ is set to 13, the appropriate value of $H$ is $\frac{K}{M} \times \ln 2 \approx 9.1$, because the value of $H$ is an integer, we set it to $\lfloor \frac{K}{M} \times \ln 2 + \frac{1}{2} \rfloor = 9$. Considering the length limit, we give a small fraction of the value of $F$ under various $\frac{K}{M}$ and $H$ combinations in Table II.

For example, if $M = 1M$ and $s = 0.001$, then from Table II, we see that if $K = 16M$, the value of $H$ should be 11, therefore $F \approx 0.000459$, and the upper bound of $E(\#FPs)$ ,calculated by Inequation 7, is $1M \times 0.000459 + 1/0.001 = 1459$. If $K = 17M$, the corresponding values of $H$ and $F$ are 12 and 0.000284, therefore $E(\#FPs) = 1284$. In fact, a much tighter bound of $E(\#FPs)$ can be derived from Inequation 8 once the value of $K$ and $H$ are confirmed.

**Analysis of $E(\#FPs)$.** Our goal is to calculate the maximum possible value of $E(\#FPs)$ for different values of $n$ when the values of $H$ and $K$ are confirmed. Let $F(n) = (M - n)(1 - (1 - \frac{1}{K})^{Hn})^H$, and from Inequation 8, we have $E(\#FPs) \leq F(n) + \lfloor \frac{1}{s} \rfloor$, in which the value of $\lfloor \frac{1}{s} \rfloor$ is confirmed once the value of $s$ is confirmed, therefore our task is to calculate the maximum value of $F(n)$. Let $t = (1 - \frac{1}{K})^H (0 < t < 1)$, and we have:

$$F(n) = (M - n)(1 - t^n)^H \tag{16}$$

Obviously, when $0 < n < M$, $F(n) > 0$, and we have:

$$F(n) = e^{ln(M-n) + Hln(1-t^n)} \quad (0 < n < M) \tag{17}$$

Let $f(n) = ln(M-n) + Hln(1-t^n)$, therefore $F(n) = e^{f(n)}$, and our task is transformed to caculate the maximum value of $f(n)$. The first derivative of $f(n)$ with respect to $n$, denoted as $f'(n)$, is:

$$f'(n) = \frac{1}{n - M} + \frac{Ht^n lnt}{t^n - 1} \quad (0 < n < M) \tag{18}$$

$$= \frac{1 - t^{-n} + nHlnt - MHlnt}{(n - M)(1 - t^{-n})} \tag{19}$$

Obviously, when $0 < n < M$, $(n - M)(1 - t^{-n}) > 0$. Let $g(n) = 1 - t^{-n} + nHlnt - MHlnt$, and the first derivative of $g(n)$ with respect to $n$, denoted as $g'(n)$, is:

$$g'(n) = t^{-n}lnt + Hlnt \quad (0 < n < M) \tag{20}$$

Obviously, $g'(n) < 0$, which means as $n$ increases, $g(n)$ decreases. In other words, we can get the value of $n$ which

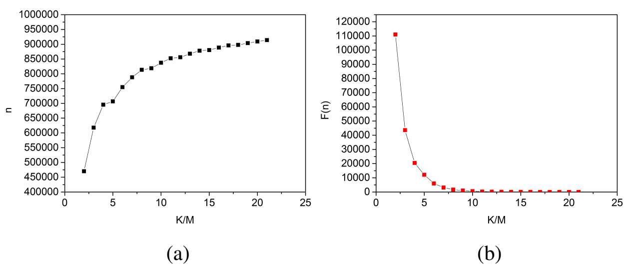



Fig. 3: The maximum value of $F(n)$ and the corresponding value of $n$ when $M = 1M$ - Varying $\frac{K}{M}$.

makes the value of $f(n)$ maximum by solving the equation $g(n) = 0$.

$$t^{-n} - 1 = nHlnt - MHlnt \quad (0 < n < M) \tag{21}$$

When $M = 1M, K = 16M$ and $H = 11$, we have:

$$(1 - \frac{1}{16 \times 10^6})^{-n} - 1 = 11 \times ln((1 - \frac{1}{16 \times 10^6}))n - 10^6 \times 11 \times ln((1 - \frac{1}{16 \times 10^6}))(0 < n < 10^6) \tag{22}$$

The approximate integer solution of Equation 22, solved by MATLAB, is 888640, and the corresponding value of $F(n)$ is 20, which means the maximum possible expected value of the false positives of *BF* is 20 which is much smaller than the value calculated before, therefore $E(\#FPs) \leq 20 + 1/0.001 = 1020$ when $s = 0.001$. Fig. 3 shows the maximum value of $F(n)$ and the corresponding value of $n$ when $M = 1M$ varying the the value of $\frac{K}{M}$, and we set $H = \lfloor \frac{K}{M} \times ln2 + \frac{1}{2} \rfloor$ as discussed before. Fig .3a shows that the value of $n$ increases as the value of $\frac{K}{M}$ increases. From Fig. 3b, we observe that the value of $F(n)$ decreases rapidly as the value of $\frac{K}{M}$ increases, however, the larger the values of $K$ and $H$ are, the more space and update time *BFSS* will consume. Taking the number of *FPs*, update time and space available into consideration, we can choose the appropriate values of $K$ and $H$.

*Theorem 4:* The space complexity of *BFSS* is $O(\lambda M + min(\lceil \frac{1}{\epsilon} \rceil, M))$, where $\lambda \in N^*$, the value of which is discussed detaily above.

*Proof:* The space complexity of *BFSS* includes two part: i) the size of *BF*, i.e. $K$. ii) the number of counters used in $D$, i.e. $m$. In order to get the minimum value of $E(\#FPs)$, the value of $K$ should be multiple times of the value of $M$, i.e. $K = \lambda M(\lambda \in N^*)$. From Algorithm 2, we have $m \leq min(\lceil \frac{1}{\epsilon} \rceil, M)$. So the space complexity of *BFSS* is $O(\lambda M + min(\lceil \frac{1}{\epsilon} \rceil, M))$. ■

In conclusion, there exists a trade off between the number of *FPs* and the space consumption of *BFSS*. Theoretically, once $M$ is confirmed, the larger the value of $\lambda$ is, the larger the value of $K$ will be, and the smaller the value of $E(\#FPs)$ will be.

Furthermore, consider a naive method to solve *s-BLFE*: we simply allocate each item in $A$ a counter, and update the corresponding counter for each item in $S$, when a query comes, we just output the items with $f_S(e) \leq \lfloor sN \rfloor$. Obviously the method can maintain the low-frequency items precisely, and the space complexity of the method is $O(M)$. *BFSS* has no advantage over the naive method in space complexity considering big O though, *BFSS* needs not to store the exact value or the fingerprint of each item which may consume much space especially when the value is long string, like URL. The experimental results indicate *BFSS* is much more space efficient.

*4) **Time complexity**:* In this section, we will discuss the time complexity of *BFSS*.

*Theorem 5:* Processing each item needs $O(1)$ time, independent of $N$.

*Proof:* From Algorithm 1, we know the processing time for each item has two parts: i) the time spent hasing each item into *BF*. ii) the time spent updating the corresponding counter in $D$. Obviously, the time spent hashing each item into *BF* is constant because $H$ is constant. Consider two cases in updating the corresponding counter in $D$: i) item $e$ is monitored in $D$, and we only have to update the corresponding counter. ii) item $e$ is not monitored in $D$, and we have to locate the counter with the minimum $f(e)$ first, then update it. Obviously, the time spent for any case is constant because $\lceil \frac{1}{\epsilon} \rceil$ is constant. Therefore, processing each item needs $O(1)$ time. ■

*5) **Challenges to BFSS**:* In this section, we will identify the existing problems in *BFSS*.

From the analysis in section III-C3, we observe that $K$ should increase monotonically with $M$ in order to keep a low rate of *FPs*, however, what if the space available is limited and small? The precision of *BFSS* will be very low in this case.

Inspired by the algorithm proposed in [11], we present the *SBFSS* algorithm, which can find low-frequency items over data streams in limited and small space with a few *FPs* together with theoretically bounded number of *FNs*.

### *D. The BFSS* Algorithm*

In this section, we will introduce *BFSS**, which approximately estimates the distribution of a data stream.

The goal of *BFSS** is to estimate the percentage of the number of the items whose frequency is above a user-specified support, denoted as $p_1$, and the percentage of the total frequencies of these items, denoted as $p_2$.

*BFSS** and *BFSS* share the same update process, and the query process of *BFSS** is described in Algorithm 3. For each item $e$ in $A$, we first check whether it is in *BF*, if the item is in *BF*, we increment the counter $n$, and if the item appears in $D$ and $f(e) > \lfloor sN \rfloor + \Delta(A[i])$, we increment $n'$ and add $f(e) - \Delta(e)$ to $f$. At last, we set $p'_1 = n'_s/n'$ as the approximate value of $p_1$ and $p'_2 = f'/N$ as the approximate value of $p_2$.

### *E. Analysis of BFSS**

In this section, we will give theoretical analysis of $p_1$ and $p_2$. Obviously, the time and space complexity of *BFSS** are the same as *BFSS*, and we don't discuss here.

**Algorithm 3** BFSS* Query Algorithm

**Input:** *BF*, $D$, $s$, $A$, $N$, $M$

**Output:** The percentage of the items whose frequency is above $\lfloor sN \rfloor$ and the percentage of the total frequencies of these items

$flag = true, n'_s = 0, n' = 0, f'_s = 0, p'_1 = 0, p'_2 = 0$;{*flag*: indicate whether an item is in *BF*, $n$: the number of distinct items in $S$, $n'_s$: the estimated number of the items whose frequency is above $\lfloor sN \rfloor$, $f'_s$: the estimated total frequencies of the items whose frequency is above $\lfloor sN \rfloor$, $p'_1$: the estimated percentage of the items whose frequency is above $\lfloor sN \rfloor$, $p'_2$: the estimated percentage of the total frequencies of these items.}<br>
**for** $i = 0$ to $M - 1$ **do**<br>
  $flag = true$;<br>
  **for** $j = 1$ to $H$ **do**<br>
    **if** $BF[h_j(A[i])] == 0$ **then**<br>
      $flag = false$;<br>
      break;<br>
    **end if**<br>
  **end for**<br>
  **if** $flag == true$ **then**<br>
    $n' = n' + 1$;<br>
    **if** $A[i]$ is monitored in $D$ **then**<br>
      **if** $f(A[i]) > \lfloor sN \rfloor + \Delta(A[i])$ **then**<br>
        $n'_s = n'_s + 1$;<br>
        $f'_s = f'_s + (f(A[i]) - \Delta(A[i]))$;<br>
      **end if**<br>
    **end if**<br>
  **end if**<br>
**end for**<br>
$p'_1 = n'_s/n', p'_2 = f'_s/N$;<br>
Output $p'_1, p'_2$;

*1) **Analysis of** $p_1$:* From the defination of $p_1$, we know that $p_1 = \frac{n_s}{n}$, in which $n_s$ denotes the exact number of the items whose frequency is above $s$. From Lemma 3 and Lemma 5, we observe two facts: i) any item with $f_s(e) > \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ must satisfy $f(e) > \lfloor sN \rfloor + \Delta(e)$. ii) any item with $f_s(e) \leq \lfloor sN \rfloor$ must satisfy $f(e) \leq \lfloor sN \rfloor + \Delta(e)$. From the two facts, we know that only items with $\lfloor sN \rfloor < f_s(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ are likely to be neglected, which means $n_s \geq n'_s$. Let $\Delta(n_s) = n_s - n'_s$, and the maximum value of $\Delta(n_s)$ is the number of the items with $\lfloor sN \rfloor < f_s(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$. Obviously, for any data stream, $\Delta(n_s) \leq \lfloor \frac{1}{s} \rfloor$, which is a very loose bound, and for data streams under Pareto distributions, we have $E(\Delta(n_s)) \leq ((\frac{f_S(e_m)}{\lfloor sN \rfloor})^{\alpha} - (\frac{f_S(e_m)}{\lfloor sN \rfloor + \lfloor \epsilon N \rfloor})^{\alpha})n$, the proof of which can be found in the proof of Theorem 3.

From Algorithm 3, we can get $n' \geq n$ because *BF* causes no false negatives but only false positives. Let $\Delta(n) = n' - n$, and the value of $\Delta(n)$ equals to the number of the false positives caused by *BF*. Therefore, we have $E(\Delta(n)) = (M - n)(1 - (1 - \frac{1}{K})^{Hn})^H < M(1 - (1 - \frac{1}{K})^{HM})^H$, the proof of which can be found in the proof of Theorem 2.

From above, $p'_1 = \frac{n_s - \Delta(n_s)}{n + \Delta(n)} \leq p_1$, and we can theoretically get the upper bound of $E(\Delta(n_s))$ for data streams under Pareto distributions and the upper bound of $E(\Delta(n))$ for any data streams. Furthermore, we can minimum $E(\Delta(n))$ by selecting appropriate parameters, which has been discussed before. Experimental results in [24], [25] show that *Space Saving* has a very high precision (almost 1), which means the number of the items with $\lfloor sN \rfloor - \lfloor \epsilon N \rfloor < f_s(e) \leq \lfloor sN \rfloor$ wrongly output is very small, and if we set $s = s + \epsilon$, then the number of the items with $\lfloor sN \rfloor < f_s(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ wrongly output is vary small too, in other words, the value of $\Delta(n_s)$ is very small. In fact, the experimental results also show that $p'_1$ is nearly the same as $p_1$.

*2) **Analysis of** $p_2$:* From the defination of $p_2$, we can get $p_2 = \frac{f_s}{N}$, in which $f_s$ denotes the exact total frequencies of the items whose frequency is above $\lfloor sN \rfloor$. Obviously, $f_s \geq f'_s$ because we may neglect some items with $\lfloor sN \rfloor < f_s(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ just like the analysis of $p_1$, besides, for the items with $f(e) > \lfloor sN \rfloor + \Delta(e)$, we just add $(f(e) - \Delta(e))(\leq f_s(e))$ to $f'_s$. Let $\Delta(f_s) = f_s - f'_s$, from above, $\Delta(f_s)$ consists of two parts: i) the total frequencies of the items with $\lfloor sN \rfloor < f_s(e) \leq \lfloor sN \rfloor + \lfloor \epsilon N \rfloor$ wrongly neglected, denoted as $\Delta_1(f_s)$. ii) the total missing frequencies of the items with $f(e) > \lfloor sN \rfloor + \Delta(e)$, denoted as $\Delta_2(f_s)$. For $\Delta_1(f_s)$, obviously, we have $\Delta_1(f_s) \leq \Delta(n_s) \times (\lfloor sN \rfloor + \lfloor \epsilon N \rfloor)$, in which $\Delta(n_s)$, for *Space Saving*, is very small (nearly 0). For $\Delta_2(f_s)$, we have $\Delta_2(f_s) \leq \lceil \frac{1}{s} \rceil \times \lfloor \epsilon N \rfloor$ because there are at most $\lceil \frac{1}{s} \rceil$ items with $f(e) > \lfloor sN \rfloor + \Delta(e)$, then from Lemma 5 and Lemma 3, we have $f_s(e) - (f(e) - \Delta(e)) \leq \Delta(e) \leq \lfloor \epsilon N \rfloor$.

From above, $p'_2 = \frac{f_s - (\Delta_1(f_s) + \Delta_2(f_s))}{N} \leq p_2$. Let $\Delta(p_2) = p_2 - p'_2$, and we have $\Delta(p_2) \leq \frac{\Delta(n_s) \times (\lfloor sN \rfloor + \lfloor \epsilon N \rfloor) + \lceil \frac{1}{s} \rceil \times \lfloor \epsilon N \rfloor}{N}$. In fact, the experimental results show that $p'_2$ is a little smaller than $p_2$.

### F. The SBFSS Algorithm

In this section, we will introduce *SBFSS*, which modifies *BFSS* and can find low-frequency items in data streams using limited and small space.

The major difference between *BFSS* and *SBFSS* is the Bloom filter we use. Concretely speaking, *BF* is a regular Bloom filter, while *SBF* change bits in *BF* into cells, each consisting of one or more bits. In addition, from the explaination above, the size of *BF*, i.e. $K$, should increase monotonically with $M$, however, there can be no linear relationship between the size of *SBF*, i.e. $K'$, and $M$, and $K'$ can be any integer predefined by users. Defination 1 is the detailed defination of *SBF*.

*Definition 1: SBF* is defined as an array of integer $SBF[1], SBF[2], ..., SBF[k]$ whose minimum value is 0 and maximum value is $max$. Each item of the array is allocated $d$ bits. The relation between $max$ and $d$ is then $max = 2^d - 1$. Compared to bits in a regular Bloom lter, each item of the *SBF* is called a cell.

Like *BFSS*, *SBFSS* consists of two phases: updating and quering. Algorithm 4 and Algorithm 5 give the detailed descriptions of them. Due to the length limit, no more tautology here.

From Algorithm 4, we observe that there are two kinds of operation on *SBF*. First randomly decrement $P$ cells by 1 so as to make room for fresh items; then set the same $H$ cells as in the update process to $max$.

**Algorithm 4** SBFSS Update Algorithm

**Input:** Stream $S$, support threshold $s$

The form of the counters in $D$ is $(e, f(e), \Delta(e))$
**for** $i = 0$ to $k - 1$ **do**
  $SBF[i] = 0$
**end for**
**for** each item $e$ of stream $S$ **do**
  Select $P$ different cells uniformly at random $SBF[j_1]...SBF[j_P]$, $P \in \{1, ..., k\}$
  **for** each cell $SBF[j] \in \{SBF[j_1]...SBF[j_P]\}$ **do**
    **if** $SBF[j] \geq 1$ **then**
      $SBF[j] = SBF[j] - 1$
    **end if**
  **end for**
  **for** $i = 1$ to $H$ **do**
    $SBF[h_i(e)] = max$
  **end for**
  **if** $e$ is monitored in $D$ **then**
    $f(e) = f(e) + 1$;
  **else if** $m < C$ **then**
    Assign a new counter $(e, 1, 0)$ to it
    $m = m + 1$
  **else**
    Let $e_m$ be the item with least hits, $min$
    Replace $e_m$ with $e$ in $D$
    $f(e) = min + 1, \Delta(e) = min$
  **end if**
  $N = N + 1$;
**end for**

**Algorithm 5** SBFSS Query Algorithm

**Input:** *SBF*, $D$, $s$, $A$, $N$, $M$
**Output:** low-frequency items with threshold $s$

$flag = true$; { *flag*: indicate whether an item is in *SBF* or not}
**for** $i = 0$ to $M - 1$ **do**
  $flag = true$
  **for** $j = 1$ to $H$ **do**
    **if** $SBF[h_j(A[i])] == 0$ **then**
      $flag = false$
      break;
    **end if**
  **end for**
  **if** $flag == true$ **then**
    **if** $A[i]$ is monitored in $D$ **then**
      **if** $f(A[i]) \leq \lfloor sN \rfloor + \Delta(A[i])$ **then**
        output $A[i]$ as a low-frequency item
      **end if**
    **else**
      output $A[i]$ as a low-frequency item
    **end if**
  **end if**
**end for**

Intuitively, due to the random deletion operation, *SBF* does not exceed its capacity in a data stream scenario, however, false negatives may come up as well.

### G. Analysis of SBFSS

Like *BFSS*, we will give theoretical analysis of *SBFSS* in this section, including the equivalence of *SBF'* (the *SBF* used in [11]) and *SBF*, *FNs*, *FPs*, parameters setting, and time complexity.

*1)* ***Analysis of the equivalence of SBF' and SBF****:* *SBF'* is used to detect duplicates over data streams, in fact, from Algorithm 4 and Algorithm 5, we can observe that *SBF* is used to filter out the items that never appeared in $S$, and we check whether the item in $A$ is duplicated one by one, if the item is a duplicate, we just claim it appeared in $S$, in this sense, *SBF'* and *SBF* share the same function. So the properties of *SBF'*, including the probability of a false positive and a false negative of *SBF'*, are fully applicable to *SBF*.

*2)* ***Analysis of FPs****:* In this section, we will give a theoretically upper bound of the expectation of the number of *FPs* in the ouput of *SBFSS*.

The detailed proof of Lemma 8 can be found in [11]. Because of the limitation of length, no more tautology here.

*Lemma 8:* The probability of a false positive (FP) of *SBF* is no greater than :

$$(1 - (\frac{1}{1 + \frac{1}{P(1/H - 1/m)}})^{max})^H \tag{23}$$

*Theorem 6:* Assuming no specific data distribution, the expectation of the number of *FPs* in the output of *SBFSS*, denoted as $E'(\#FPs)$, satisfies:

$$E'(\#FPs) < M \min\{(1 - (\frac{1}{1 + \frac{1}{P(1/H - 1/m)}})^{max})^H, \quad (1 - (1 - \frac{1}{k})^{HM})^H\} + \lceil \frac{1}{s} \rceil \tag{24}$$

*Proof:* From Fig. 4, we notice that the *FPs* of *SBFSS* has two parts: i) the false positives of *SBF*, i.e. $II$; ii) the items with $f_S(e) > \lfloor sN \rfloor$ wrongly output, i.e. $VI$. In fact, $V$ is the only difference between the *FPs* of *SBFSS* and the *FPs* of *BFSS*, for *BFSS*, the items in $V$ will be output, however, due to the false negatives of *SBF* in *SBFSS*, i.e. $IV + V + VIII$, the items in $V$ will be neglected by *SBFSS*, in this sense, the false negatives of *SBF* inversely reduce the number of *FPs* of *SBFSS*.

In fact, the posibility of a false positive of *SBF* must be smaller than that of *BF* under the same conditions because *SBF* brings in decrement operation. For the first case, we can get the upper bound of the expectation of the number of the false positives of *SBF* through Lemma 7 and Lemma 8. For the second case, it is clearly to see from Fig. 4 that the maximum of the area of $VI$ is $\lceil \frac{1}{s} \rceil$, which is the area of the square surrounded by dotted lines, and obviously it is a loose bound. The residual derivation is obvious and exactly the same as Theorem 2, and no more tautology here. ■

Like *BFSS*, a tighter bound of the area of $VI$ can be obtained if we assume the distribution of $S$, and the bound is $(\frac{f_S(e_m)}{\lfloor sN \rfloor})^\alpha (1 - \frac{1}{2^\alpha})M$, the derivation of which is exactly the same as E.q. 10, if the distribution is Pareto. In addition, due to the exsitence of $V$, the number of the elemens with $f_S(e) > \lfloor sN \rfloor$ wrongly output is smaller than that of *BFSS* regardless of the data distribution of $S$.

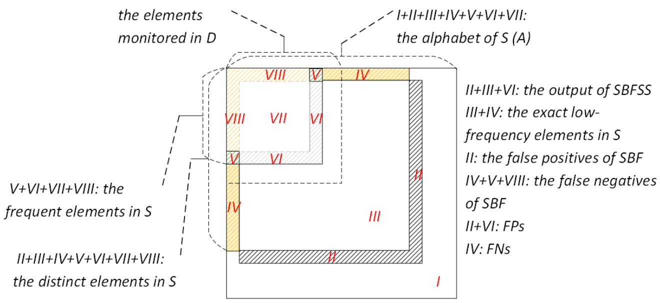



Fig. 4: The schematic diagram of the analysis of *SBFSS*

*3) **Analysis of FNs**:* In this section, unlike *BFSS*, we will give a theoretically upper bound of the expectation of the number of *FNs* in the ouput of *SBFSS*.

From Fig. 4, we can see that the *FNs* of *SBFSS*, $IV$, is a portion of the false negatives of *SBF*, $IV + V + VIII$. Therefore the upper bound of the number of *FNs* in the output of *SBFSS* is the number of the false negatives of *SBF*.

First, let us consider the probability of a false negative of *SBF* which is an error when an item in $S$ wrongly filtered out by *SBF*. Therefore only the items in $S$ can generate false negatives, obviously the number of false negatives is related to the input data distribution.

Suppose an item $A[i]$ appearing in $S$ whose last appearence in $S$ before a query comes is $e_{i-\delta_i}$, where $\delta_i$ represents the number of items in $S$ during the time between the last appearance of $A[i]$ and a query, is hashed into $H$ cells, $SBF[C_{i1}], ..., SBF[C_{iH}]$. A false negative happens if any of those $H$ cells is decremented to 0 during the $\delta_i$ iterations when a query comes. Let $PR_0(\delta_i, p'_{ij})$ be the probability that cell $C_{ij}(j = 1...H)$ is decremented to 0 within the $\delta_i$ iterations, and $A_l$ denotes the event that within the $N$ iterations the most recent setting operation applied to the cell occurs at iteration $N - l$, and $A'_N$ denotes the event that cell has never been set within the whole $N$ iterations. $PR_0(\delta_i, p'_{ij})$ can be computed as:

$$PR_0(\delta_i, p'_{ij}) = \sum_{l=max}^{\delta_i - 1}[Pr(SBF_{\delta_i} = 0|A_l)Pr(A_l)] \\ +Pr(SBF_{\delta_i} = 0|A_{\delta_i})Pr(A'_{\delta_i}) \tag{25}$$

where

$$Pr(SBF_{\delta_i} = 0|A_l) = \sum_{j=max}^{l} \binom{l}{j} p^j (1-p)^{l-j} \tag{26}$$

$$Pr(A_l) = (1 - p'_{ij})^l p'_{ij} \tag{27}$$

$$Pr(SBF_{\delta_i} = 0|A'_{\delta_i}) = \sum_{j=max}^{\delta_i} \binom{\delta_i}{j} p^j (1-p)^{\delta_i - j} \tag{28}$$

$$Pr(A'_{\delta_i}) = (1 - p'_{ij})^{\delta_i} \tag{29}$$

Based on E.q 25, the expected value of the false negatives of *SBF*, $E(\#FN)$, can be easily derived, and the tailed derivation of E.q 25 and Lemma 9 can be found in [11], therefore no more tautology here.

*Lemma 9:* There must be an "average" $\widehat{\delta}$ and an "average" $\widehat{p'}$ such that:

$$E(\#FN) = n[1 - (1 - PR_0(\widehat{\delta}, \widehat{p'}))^H] \tag{30}$$

*Theorem 7:* Assuming no specific data distribution, the expectation of the number of *FNs* in the output of *SBFSS*, denoted as $E'(\#FNs)$, satisfies:

$$E'(\#FNs) \leq M[1 - (1 - PR_0(\widehat{\delta}, \widehat{p'}))^H] \tag{31}$$

*Proof:* Like *FPs*, two kinds of items contribute to *FNs*: i) the elemtents with $0 < f_S(e) \leq \lfloor sN \rfloor$ wrongly filtered out by *SBF*; ii) the elemtents with $f_S(e) \leq \lfloor sN \rfloor$ monitored in $D$ wrongly neglected. In fact, the items with $f_S(e) \leq \lfloor sN \rfloor$ must satisfy $f(e) \leq \lfloor sN \rfloor + \Delta(e)$, which means no items with $f_S(e) \leq \lfloor sN \rfloor$ monitored in $D$ wrongly neglected. For the first case, we assume that the false negatives are all items with $0 < f_S(e) \leq \lfloor sN \rfloor$, i.e. neglecting $V + VIII$ in Fig. 4, which is the worst case, and in this way we have $E(\#FN) = E(\#FNs) = n[1 - (1 - PR_0(\widehat{\delta}, \widehat{p'}))^H] \leq M[1 - (1 - PR_0(\widehat{\delta}, \widehat{p'}))^H]$. ■

From E.q 24, we observe that the upper bound of $E'(\#FPs)$ is completely related to *SBF* once the parameter $s$ is confirmed, and the upper bound of $E'(\#FNs)$ is totally related to *SBF* which can be observed from E.q 31. In this way, we will give some analysis of how to choose appropriate parameters of *SBF* next.

*4) **Parameters setting**:* In this section, we will discuss how to choose appropriate parameters of *SBF* in *SBFSS*.

For *SBFSS*, users will specify $s$, $\#FPs$ and memory limit $L$. The memory consumption of *SBFSS* mainly contains two parts: *SBF* and $D$. From Algorithm 4, we know that there are at most $\lceil \frac{1}{s} \rceil$ counters in $D$, and in this way we can estimate the memory available for *SBF*, and through E.q 24, we can estimate the FP rate, in fact, the FP rate can be a little larger than the caculated one beacause only a part of items cause false positives. Therefore our goal is to choose a combination of $max, H$ and $P$ that minimizes the number of FNs under the condition that the FP rate is within a confirmed threshold and a fix amount of space.

In fact, a detailed discussion of how to choose these parameters has been given in [11], which can concluded as: A formula can be obtained to compute the value of $P$ from other parameters provided that $max$ and $H$ have been chosen already; then find the optimal values of $H$ for each case of $max(1, 3, 7)$ by trying limited number($\leq 10$) of values of $H$ on the FN rate formulas; Last, $max$ can be set empirically, and specially in the case that no prior knowledge of the input data is available, $max$ was suggested to be 1;

*5) **Time complexity**:* In this section, we will analyze the time complexity of *SBFSS*.

For *SBFSS*, the amount of space has been confirmed, so we just focus on time complexity.

*Theorem 8:* For *SBFSS*, Processing each data stream item needs $O(1)$ time, independent of the size of space and the stream.

*Proof:* Like *BFSS*, the processing time for each item also consists two parts: i) the time spent updating *SBF*; ii) the time

TABLE III: The maximum value of $F(n)$ and the corresponding value of $n$ under various $K$ and $H$ combinations when $M$=89753.

| $K$ | $H$ | $n$ | $F(n)$ | $K$ | $H$ | $n$ | $F(n)$ |
|---|---|---|---|---|---|---|---|
| 179506 | 1 | 42191 | 9962 | 897530 | 7 | 75154 | 49 |
| 269259 | 2 | 55448 | 3909 | 987283 | 8 | 76505 | 28 |
| 359012 | 3 | 62432 | 1835 | 1077036 | 8 | 76809 | 17 |
| 448765 | 3 | 63423 | 1087 | 1166789 | 9 | 77890 | 9 |
| 538518 | 4 | 67742 | 538 | 1256542 | 10 | 78794 | 5 |
| 628271 | 5 | 70752 | 281 | 1346295 | 10 | 79006 | 3 |
| 718024 | 6 | 72992 | 152 | 1436048 | 11 | 79758 | 2 |
| 807777 | 6 | 73467 | 90 | 1525801 | 12 | 80406 | 1 |

spent updating $D$. In fact, from Algorithm 1 and Algorithm 4, we can see that *BFSS* and *SBFSS* share the same operations on $D$, so for *SBFSS*, the time spent updating $D$ is constant once $s$ is confirmed, which means the size of space has no impact on it. For the first part, a detailed explaination has been given in [11] to prove that the time spent updating *SBF* is O(1), independent of the size of space and the stream. Therefore the time complexity of *SBFSS* is O(1). ■

## IV. Experiments

In this section, we first describe our data sets and the implementation details of *BFSS* and *SBFSS*, then we tested the performance of *BFSS* under different parameter settings on both real and synthetic data sets, then we compared the results against the method *nCount* and the method *rCount*. We were insterested in the *recall*, the number of correct items found as a percentage of the number of correct items, and the *precision*, the number of correct items found as a percentage of the entire output, then we calculated $F_1$. We also measured the space used by *BFSS*, *nCount* and *rCount*, which is essential to handle data streams, however, due to the page limit, we can not show the results of synthetic data here. For *SBFSS*, we compared the recall and precision against the method *lCount* on real data in different size of limited space. The detailed implementations of *nCount*, *rCount* and *lCount* will be given later. Last, we give a summarization of the experimental results. All the algorithms were compiled using the same compiler, and were run on a AMD Opteron(tm) 2.20GHz PC, with 64GB RAM, and 1.8TB Hard disk.

### A. Data Sets

**Real Data.** For real data experiments, we used the dataset[5] which consists of all the requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. We extracted $7 \times 10^7$ URLs in total, and the size of the alphabet is 89753.

**Synthetic Data.** We generated several synthetic Zipfian data sets with the Zipf parameter varying from 0.5, which is very slightly uniform, to 3.0, which is highly skewed, with a fixed increment of 0.5. The size of each data set, N, is $10^7$, and the size of the alphabet is $10^6$.

### B. Implementation Issues

***BFSS* Implementation.** It is simple and straight forward to implement *BFSS*: 1) hash each incoming stream item into $H$ numbers, and set the corresponding $H$ bits to 1. 2) we maintain a min heap to index each counter in $D$, if there exists an empty counter, we just insert a new counter into the heap, otherwise we replace the top counter of the heap, at last we readjust the heap. 3) when a query comes, we just do as Algorithm 2 indicates.

Table III lists the maximum value of $F(n)$ and the corresponding value of $n$ under various $K$ and $H$ combinations when $M$=89753, in which $H = \lfloor \frac{K}{M} \times ln2 + \frac{1}{2} \rfloor$. The maximum value of $F(n)$ is 3909 when $\frac{K}{M} = 3 (K = 269259)$ and $H = 2$, and the upper bound of $E(\#FPs)$ can be confirmed once the value of $s$ is confirmed, which can be observed from Inequation 8, for example $E(\#FPs) \leq 4909$ when $s = 0.001$. Taking $E(\#FPs)$, update time and space consumption into consideration, we set $K = 270000, 400000, 800000$ separately and $H = 2$, and the corresponding maximum value of $F(n)$ is 3892, 2014 and 579 which decreases as $K$ increases, and this can be easily observed from the expression of $F(n)$.

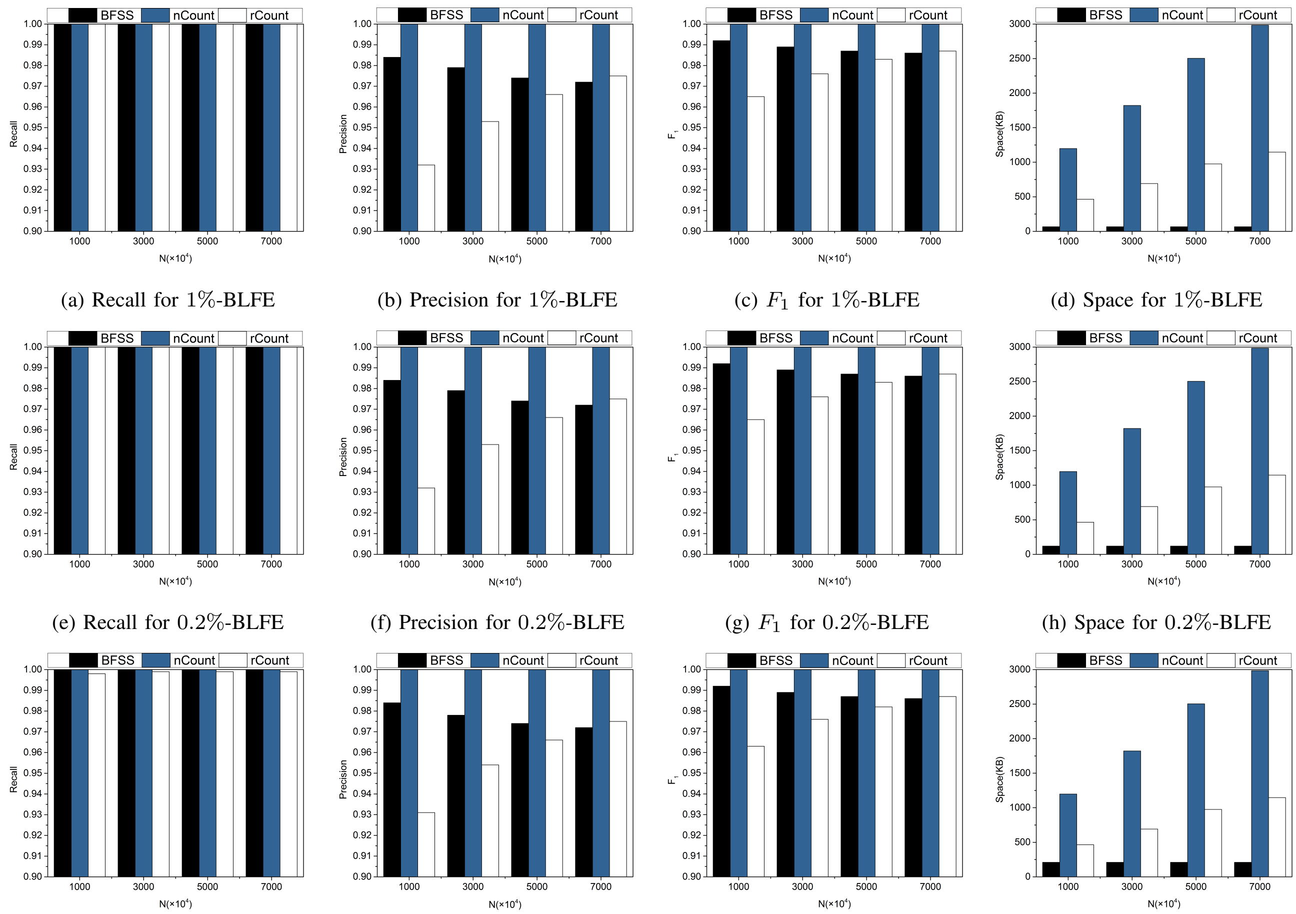
***SBFSS* Implementation.** The only difference between *BFSS* implementation and *SBFSS* implementation is the implementation of *BF* and *SBF*. For *SBF*, we generate a random number in each iteration, decrement the corresponding cell and $(P-1)$ cells adjacent to it by 1; this process is faster than generating $P$ random numbers for each item; although the processes of picking the $P$ cells are not independent, each cell has a probability of $P/k$ for being picked at each iteration. Our analysis still holds. Then we hash each incoming stream item to $H$ numbers, and set the corresponding $H$ cells to max. Taking the number of *FNs*, the number of *FPs*, update time and limited space we can use into consideration, we set $max = 3, H = 4$ and $P = 1$.

***nCount* Implementation.** The basic idea of *nCount* is to allocate each distinct item in $S$ a counter, we use a hash table to index these counters to speedup updating, and it is time consuming if we use a linked list or an array. When an item comes, we increment the corresponding counter using hash table. When a query comes, we traverse the hash table and output the low-frequency items.

***rCount* Implementation.** *rCount* shares a similar idea with *nCount*, however, unlike *nCount*, *rCount* uses Rabin's method [29] to hash each item in $S$ to a 64-bit fingerprint, or it will consume too much space especially when the items are URLs. With this fingerprinting technique, there is a small chance that two dierent URLs are mapped to the same fingerprint. When an item comes, we first calculate its fingerprint and then increment the corresponding counter using hash table. When a query comes, we traverse the alphabet $A$ and calculate the fingerprint of each item in $A$, then we check the corresponding counter and output it if it is a low-frequency item.

***lCount* Implementation.** *lCount* maintains a limited and fixed space, like *rCount*, *lCount* uses Rabin's method in order to adjust the space consumption through the maximum number of counters *lCount* can maintain. In addition, we use a max heap to index the counters in *lCount*. When an item comes, if there exists an empty counter, we insert a new counter into the heap, otherwise we replace the top counter of the heap, i.e. the counter with the maximum value, and set the value of the new counter to 1, at last we readjust the heap. When a query comes, we traverse the alphabet $A$ and calculate the fingerprint

[5] http://ita.ee.lbl.gov/html/contrib/WorldCup.html




(a) Recall for 1%-BLFE (b) Precision for 1%-BLFE (c) $F_1$ for 1%-BLFE (d) Space for 1%-BLFE

(e) Recall for 0.2%-BLFE (f) Precision for 0.2%-BLFE (g) $F_1$ for 0.2%-BLFE (h) Space for 0.2%-BLFE

(i) Recall for 0.1%-BLFE (j) Precision for 0.1%-BLFE (k) $F_1$ for 0.1%-BLFE (l) Space for 0.1%-BLFE

Fig. 5: Performance Comparison between *BFSS*, *nCount* and *rCount* Using Real Data - Varying $s, \epsilon$ and $N$

TABLE IV: Performance of *BFSS* Using Real Data - Varying $s$, $\epsilon$, $K$ and $N$

| $s$ | $\epsilon$ | $K$ | Space | Precision | | | |
|---|---|---|---|---|---|---|---|
| | | | | $N = 10^7$ | $N = 3 \times 10^7$ | $N = 5 \times 10^7$ | $N = 7 \times 10^7$ |
| 1% | 1% | 270000 | $49KB$ | 0.970 | 0.959 | 0.949 | 0.944 |
| | | 400000 | $66KB$ | 0.984 | 0.979 | 0.974 | 0.972 |
| | | 800000 | $116KB$ | 0.996 | 0.994 | 0.992 | 0.992 |
| 1% | 0.1% | 270000 | $193KB$ | 0.970 | 0.959 | 0.949 | 0.944 |
| | | 400000 | $210KB$ | 0.984 | 0.979 | 0.974 | 0.972 |
| | | 800000 | $260KB$ | 0.996 | 0.994 | 0.993 | 0.992 |
| 0.2% | 0.2% | 270000 | $103KB$ | 0.969 | 0.959 | 0.949 | 0.943 |
| | | 400000 | $120KB$ | 0.984 | 0.978 | 0.974 | 0.972 |
| | | 800000 | $170KB$ | 0.996 | 0.994 | 0.993 | 0.992 |
| 0.2% | 0.1% | 270000 | $193KB$ | 0.969 | 0.959 | 0.949 | 0.943 |
| | | 400000 | $210KB$ | 0.984 | 0.978 | 0.974 | 0.972 |
| | | 800000 | $260KB$ | 0.996 | 0.994 | 0.993 | 0.992 |
| 0.1% | 0.1% | 270000 | $193KB$ | 0.969 | 0.958 | 0.948 | 0.943 |
| | | 400000 | $210KB$ | 0.984 | 0.978 | 0.974 | 0.972 |
| | | 800000 | $260KB$ | 0.996 | 0.994 | 0.993 | 0.992 |

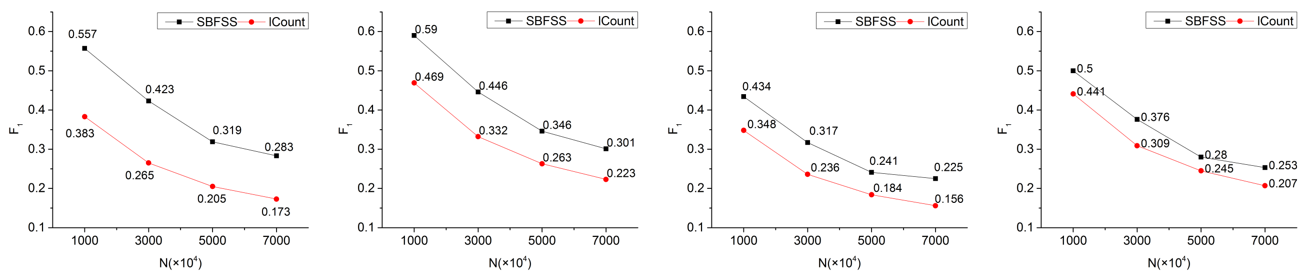
of each item in $A$, then we check if the corresponding counter exists in the heap and output it if it is a low-frequency item.

### C. Performance of BFSS

The query issued for *BFSS*, *nCount* and *rCount* was to find all items whose frequency is no more than $\frac{N}{s}(s = 1\%, 0.2\%, 0.1\%)$.

TABLE V: Performance Comparison between *SBFSS* and *lCount* Using Real Data in Limited Space - Varying $s, N$

| $s$ | Space | Precision (SBFSS / lCount) | | | | Recall (SBFSS / lCount) | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | $N = 10^7$ | $N = 3 \times 10^7$ | $N = 5 \times 10^7$ | $N = 7 \times 10^7$ | $N = 10^7$ | $N = 3 \times 10^7$ | $N = 5 \times 10^7$ | $N = 7 \times 10^7$ |
| 1% | $54KB$ | 0.999/0.916 | 1/0.919 | 0.999/0.926 | 0.999/0.949 | 0.386/0.242 | 0.268/0.155 | 0.190/0.115 | 0.165/0.095 |
| | $72KB$ | 1/0.920 | 1/0.930 | 0.999/0.940 | 0.999/0.959 | 0.419/0.315 | 0.287/0.202 | 0.209/0.153 | 0.177/0.126 |
| 0.2% | $54KB$ | 0.981/0.828 | 0.982/0.842 | 0.984/0.855 | 0.983/0.859 | 0.279/0.220 | 0.189/0.137 | 0.137/0.103 | 0.127/0.086 |
| | $72KB$ | 0.997/0.886 | 0.996/0.906 | 0.999/0.922 | 0.998/0.907 | 0.334/0.294 | 0.232/0.186 | 0.163/0.141 | 0.145/0.117 |
| 0.1% | $54KB$ | 0.981/0.828 | 0.982/0.842 | 0.984/0.855 | 0.983/0.859 | 0.279/0.220 | 0.189/0.137 | 0.137/0.103 | 0.127/0.086 |
| | $72KB$ | 0.997/0.886 | 0.996/0.906 | 0.999/0.922 | 0.998/0.907 | 0.334/0.294 | 0.232/0.186 | 0.163/0.141 | 0.145/0.117 |




(a) $s$=0.01, Space=54KB (b) $s$=0.01, Space=72KB (c) $s$=0.001, Space=54KB (d) $s$=0.001, Space=72KB

Fig. 6: Performance Comparison of $F_1$ between *SBFSS* and *lCount* - Varying $s$ and Space

The results comparing the recall, precision, $F_1$ and space used by *BFSS*, *nCount* and *rCount* are summarized in Fig. 5. The value of $N$ was varied from $10^7$ to $7 \times 10^7$, and we set $\epsilon = s, K = 400000, H = 2$. Obviously, the recalls achieved by *BFSS* and *nCount* were constant at 1 for all values of $N$ and $s$, however, due to hash collisions, *rCount* may neglect a few low-frequency items, as is clear from Fig. 5i. From Fig. 5b, Fig. 5f and Fig, 5j, we observe that the precision of *BFSS* decreased as the value of $N$ increased, in fact, as we discussed before, the value of $F(n)$ increases as the value of $n$ increases unitil $F(n)$ reaches its maximum value then the value of $F(n)$ decreases as the value of $n$ increases, which means the precision of *BFSS* showed a trend from decline to rise as the value of $N$ increased with high probability. The precision of *rCount* increased as the value of $N$ increased because only the items not appearing in $S$ may become false positives, and the number of these items decreased with the increasing of $N$. In addition, the precision of *nCount* remained unchanged at 1 because *nCount* just kept a counter for each item in $S$. Being concerned with both precision and recall, we calculated the values of $F_1$ of these algorithms based on the precisions and recalls as Fig. 5c, Fig. 5g and Fig. 5k show, and *BFSS* outperformed *rCount* except when $N = 7 \times 10^7$. The advantage of *BFSS* is evident in Fig. 5d, Fig. 5h and Fig. 5l, which show that *BFSS* achieved a greate reduction in space consumption, more specifically, when $s = 0.1\%$, *BFSS* achieved a space reduction by a factor of 5.7 to 14.2 compared with *nCount* and a space reduction by a factor of 2.2 to 5.5 compared with *rCount*, when $s = 0.2\%$, *BFSS* achieved a space reduction by a factor of 10.0 to 24.9 compared with *nCount* and a space reduction by a factor of 3.9 to 9.6 compared with *rCount*, when $s = 1\%$, *BFSS* achieved a space reduction by a factor of 18.2 to 45.2 compared with *nCount* and a space reduction by a factor of 7.0 to 17.4 compared with *rCount*. In addition, the space used by *nCount* and *rCount* increased rapidly as the value of $N$ increased since more counters were needed, however, the space used by *BFSS* was stable because the space complexity of *BFSS* is not ralated to the value of $N$, and the minor difference between the space used by *BFSS* as the value of $N$ increased existed in the space used by $D$.

Table IV shows the precision of *BFSS* with different parameter settings, and the recall of *BFSS* was constant at 1, which has been proved theoretically and practically, so we don't list here. We neglect the minor difference between the space used by *BFSS* as the value of $N$ increased. The precision increased as the value of $K$ increased, and once the values of $K$ and $N$ were confirmed the precision was almost confirmed too, which means the values of $s$ and $\epsilon$ had little affection on the precison and most of *FPs* were caused by *BF*.

### *D. Distribution estimation*

Table VI shows the estimation of the distribution of the real data when $N$ was $7 \times 10^7$, and we have two observations. First, the estimated results given by our method is precise enough, and nearly no difference exists between the estimated results and the real results, as is clear from the table. Another interesting obervation is that the most frequent one percent of the items in $W$ occupied more than seventy percent of the number of the items in $W$, which means most people were interested in a few webpages and most webpages were browsed only a few times, and from table VI, we can see that ninety-nine percent of the items in $W$ occupied no more than thirty percent of the number of the items in $W$, in fact, thirty-two percent of the items in $W$ were browsed only once, which is realy a "long tail".

### *E. Performance of SBFSS*

As mentioned before, when the total space is limited, the space left for the Bloom filter is limited as well, and it may

TABLE VI: Distribution Estimation of Real Data.

| $s$ | $> 1\%$ | $> 0.55\%$ | $> 0.1\%$ |
|---|---|---|---|
| estimated percentage of the items | 0.027% | 0.22% | 1.1% |
| estimated percentage of the frequencies | 6.0% | 36% | 72% |
| true percentage of the items | 0.028% | 0.23% | 1.1% |
| true percentage of the frequencies | 6.0% | 36% | 72% |

cause many *FPs* when the space is small. Table V shows the performance comparison between *SBFSS* and *lCount* when the space was limited. When $s$ was 0.01, *SBFSS* outperformed *lCount* in both precision and recall, and we can see that the precision of *SBFSS* was stable and constant at 1 for most values of $N$, and the precision of *lCount* increased with the increasing of $N$ the reason of which is much like *rCount*. In addition, the recalls of *SBFSS* and *lCount* decreased with $N$ since the space was limited and the information we could maintain was limited too, which directly caused the decreasing of the recalls. When $s$ was 0.001, the rules of how the precisons and recalls changed was similar to the case when $s$ was 0.01, and we can find that *SBFSS* still maintained high accuracy, however, the general performance of *SBFSS* was not as good as the case when $s$ was 0.01 because much more space could be allocated to *SBF* when $s$ was smaller. Another interesting observation is that *SBFSS* performed better when the limited space is small, which can be clearly seen from Fig. 6, which shows the comparison of the $F_1$ between *SBFSS* and *lCount*. We can observe that when $s$ was confirmed, the smaller the space available was, the larger the gap between the $F_1$ was, and this obervation proves the efficiency of *SBFSS*.

## V. Conclusion

Low-frequency items in data streams contain much useful information which can be widely used in commercial decision-making, data analysis, etc., and locating them is the first step. We present *BFSS*, an effective and space efficient algorithm that aims to solve the problem *s-BLFE* approximately. *BFSS* can output all low-frequency items and a few frequent items, the number of which can be theoretically bounded. The experimental results show the recall of *BFSS* was constant at 1 and the precision was above 0.95 using a small space. When the memory is limited and small, in which case *BFSS* can not guarantee high precision, we propose *SBFSS* which can guarantee .

The future directions of our work can be, but are not limited to: 1) output low-frequency items along with their approximate frequencies, and this can be achieved based on the data structure of the *Count-Min Sketch* algorithm as we discussed before. 2) handle sliding window queries. 3) support both insertion and deletion of items. 4) find low-frequency item sets. In fact, like frequent items, there are many work can be done towards low-frequency items, however, due to the special features of them, we still have a long and tough way to go and we just take a small step forward.

## Acknowledgment

The authors would like to thank...

## References

[1] S. Gunduz and M. Ozsu, "A web page prediction model based on click-stream tree representation of user behavior," in *SIGKDD*, 2003, pp. 535–540.

[2] J. Chen, D. DeWitt, F. Tian, and Y. Wang, "Niagaracq: A scalable continuous query system for internet databases," in *SIGMOD*, 2000, pp. 379–390.

[3] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *PVLDB*, 2002, pp. 358–369.

[4] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *MDM*, 2001, pp. 3–14.

[5] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith, "Hancock: A language for extracting signatures fromdata streams," in *SIGKDD*, 2000, pp. 9–17.

[6] E. Demaine, A. Lopez-Ortiz, and J. Munro, "Frequency estimation of internet packet streams with limited space," in *ESA*, 2002, pp. 348–360.

[7] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *STOC*, 1996, pp. 20–29.

[8] P. Haas, J. Naughton, S. Sehadri, and L. Stokes, "Sampling-based estimation of the number of distinct values of an attribute," in *PVLDB*, 1995, pp. 311–322.

[9] D. M. Kane, J. Nelson, and D. P. Woodruff, "An optimal algorithm for the distinct elements problem," in *PODS*, 2010, pp. 41–52.

[10] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," in *SIAM*, 2002, pp. 635–644.

[11] F. Deng and D. Rafiei, "Approximately detecting duplicates for streaming data using stable bloom filters," in *SIGMOD*, 2006, pp. 25–36.

[12] A. Metwally, D. Agrawal, and A. E. Abbadi, "Duplicate detection in click streams," in *WWW*, 2005, pp. 12–21.

[13] X. Lin, H. Lu, J. Xu, and J. Yu, "Continuously maintaining quantile summaries of the most recent n elements over a data stream," in *ICDE*, 2004, pp. 362–374.

[14] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surng wavelets on streams: One-pass summaries for approximate aggregate queries," in *PVLDB*, 2001, pp. 79–88.

[15] J. Gehrke, F. Korn, and D. Srivastava, "On computing correlated aggregates over continual data streams," in *SIGMOD*, 2001, pp. 13–24.

[16] P.Bose, E.Kranakis, P.Morin, and Y.Tang, "Bounds for frequency estimation of packet streams," in *SIROCCO*, 2003, pp. 33–42.

[17] G.Cormode, F.Korn, S.Muthukrishnan, and D.Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *SIGMOD*, 2004, pp. 155–166.

[18] S. G.Cormode, F.Korn, "Whats hot and whats not: Trackingmostfrequentitems dynamically," in *PODS*, 2003, pp. 296–306.

[19] Q. Huang and P. P. C. Lee, "Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams," in *INFOCOM*, 2014, pp. 1420–1428.

[20] A. Metwally, D. Agrawal, and A. E. Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *ICDT*, Jan 2005, pp. 398–412.

[21] Q. Huang and P. P. C. Lee, "Distributed top-k monitoring," in *SIGMOD*, 2003, pp. 28–39.

[22] A. Lall, V. Sekara, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *SIGMETRICS*, Jun 2006.

[23] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *PVLDB*, Aug 2002.

[24] G. Cormode and M. Hadjieleftheriou, "Methods for finding frequent items in data streams," *VLDB J.*, vol. 19, no. 1, pp. 3–20, 2010.

[25] H. Liu, Y. Lin, and J. Han, "Methods for mining frequent items in data streams: an overview," *Knowledge and Information Systems*, vol. 26, no. 1, pp. 1–30, 2011.

[26] G. Cormode and S.Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[27] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou, "Dynamically maintaining frequent items over a data stream," in *CIKM*, 2003, pp. 287–294.

[28] L. Adamic. Zipf, power-laws, and pareto - a ranking tutorial. [Online]. Available: http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html

[29] M. O. Rabin *et al.*, *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.