Finding Low-Frequency Items in Data Streams

***Abstract*—With the rapid development of information technology, low-frequency items in data streams are becoming important in specific areas, and they can be hugely valuable if we can find and take good advantage of them. However, as far as we know, there is no effective algorithm finding low-frequency items in data streams. To address the problem, we propose a one-pass algorithm, called *BFSS*, which can effectively find items in data streams 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. Experimental results show *BFSS* achieves high performance using much less memory space compared with the algorithm *nCount*, which is proposed by us and aims at finding low-frequency items in data streams accurately in a straightforward way. Furthermore, *BFSS* can be extended to *BFSS** with a little modification, which can estimate the percentage of the total frequencies of a certain proportion of items in data streams.**

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 problem about how to find low-frequency items in data streams so far, in fact, low-frequency items are becoming important in specific areas, and thay can be extremely valuable if we can find and make good use of them.

A. Motivating Examples

1) Mining long-tail keywords: Long-tail keywords are longer and more specific keyword phrases that visitors are more likely to use when they're closer to a point-of-purchase, and their searching frequencies are much lower than normal keywords. They're a little bit counter-intuitive at first, but they can be hugely valuable in *Search Engine Optimization (SEO)* if you know how to use them.

Take this example: if you're a company that sells classic furniture, the chances are that your pages are never going to appear near the top of an organic search for "furniture" because there's too much competition (this is particularly true if you're a smaller company or a startup). But if you specialize in, say, contemporary art-deco furniture, then keywords like "contemporary Art Deco-influenced semi-circle lounge" are going to reliably find those consumers looking for exactly that product.

Long-tail keywords are valuable for businesses who want their content to rank in organic Google searches, but they're potentially even more valuable for advertisers running paid search marketing campaigns. That's because when you bid on long-tailed keywords, the cost per click is inevitably lower, since there's less competition. By targeting longer, more specific long-tail keywords in your AdWords campaigns, you can get higher ad rankings on relevant searches without having to pay a premium for every click. The trick is to find a reliable, renewable source of long-tail keywords that are right for you and for your niche.

There are many tools that can generate long-tail keywords, however, most of them simply expand basic keywords and the chances are that some words are still frequent and competition is fierce as well, or some words never appeared in the past searching sreams, and chances are high that these words bring no traffic in the future, besides, some valuable long-tail keywords may even not be generated. Another problem is the frequencies of those generated long-tail keywords may change overtime, which means frequent words can become low-frequency ones and vice versa. Therefore, if we can find the related low-frequency long-tail keywords in searching streams effectively and dynamically, we can find reliable and renewable long-tail keywords, which motivates solving the problem of finding low-frequency items in data streams.

2) Mining rare demands: With the rapid development of the internet, we can shop and search whatever we are interested in online. Our demands, for example, buying a regular water glass or searching the information about a tourist attraction, are popular most of the time and they are easily satisfied. However, it is not so easy to buy embroidery stitches or find the information about a nameless village in China online, as they are rare demands.

Yusuf Mehdi, senior vice president of the Online Audience Group for Microsoft Bing, said at Search Engine Strategies[1] that 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. For Microsoft, focusing on the head instead of the "long tail" means that it returned popular queries but failed to satisfy those rare queries, which ended up yielding more sizable traffic and therefore more money for Google over the last 11-plus years.

From the example above, rare demands, in some sense, are more important than popular demands, and finding them quickly and effectively is the first step to analyzing them, which motivates our work as well.

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

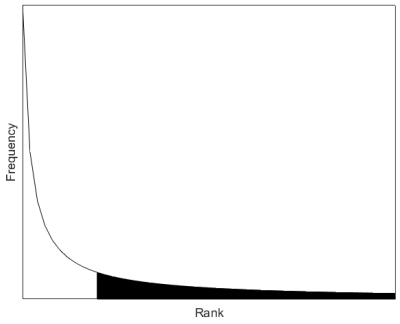


Fig. 1: Rank-frequency distribution obeying a power law form

3) Rank-frequency distribution estimation: Rank-frequency laws, such as Zipfs law and more generally power laws, have been observed in a multitude of domains ranging from the intensity of wars to the populations of major cities. As more rank-frequency laws are observed empirically and rank-frequency distributions are adopted by increasingly diverse fields such as economics and behavioral sciences the importance of practical estimating rank-frequency distributions in general similarly grows.

Fig.1 shows rank-frequency distribution obeying a power law form[2]. From Fig.1, we observe that low-frequency items form a long tail, which means low-frequency items play a important role in rank-frequency distribution estimation. Combining the information of both frequent and low-frequency items, we can have a more comprehensive estimation of the rank-frequency distribution of a data stream.

B. Our Contributions

We are the first to pose and formally define the problem *s-Bounded Low-Frequency Elements (s-BLFE)*, the definition of which will be given later.

We propose the algorithm *BFSS*, which extends the classic algorithm *Space Saving* and solves the problem *s-BLFE*. The basic idea of *BFSS* is as follows: the algorithm *Space Saving* can be used to find items whose frequency is above a user-specified support approximately, and by filtering them out we can find low-frequency items. *BFSS* outputs no false negatives and a few false positives using small memory footprints.

Given a little modification, *BFSS* can be extended to the algorithm *BFSS**, which can estimate the percentage of the total frequencies of a certain proportion of items in data streams, for example, we can approximately estimate the 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.

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

[3]"twenty" is an example, and the specific value varies

TABLE I: Major Notations Used in the Paper.

Notation	Meaning
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
ϵ	The user-specified error parameter
D	The synopsis used in *BFSS*
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*
$f_S(e)$	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
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 $\lfloor sN \rfloor$

C. Roadmap

In Section 2, we formally define the problem *s-BLFE* and highlight the related work. Our algorithms are presented and discussed in Section 3. We report the results of our experimental evaluation in Section 4. Conclusions are given in Section 5.

II. Problem Defination and Related Work

This section presents the defination of *s-BLFE* and the representative algorithms solving the problem *ϵ-Deficient Frequent Elements* [23]. 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 $\epsilon \leq s$.

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

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 wants to find all items whose frequency is no more than $0.1\%N$, then $s = 0.1\%$. The user is free to set ϵ to whatever she feels is a comfortable margin of error. As a rule of thumb, she could set ϵ to one-tenth or one-twentieth the value of s, and assume she sets $\epsilon = 0.01\%$. Therefore the algorithms solving *s-BLFE* should output all items with frequency no more than $0.1\%N$ and no item with frequency no less than $\lfloor 0.1\%N \rfloor + \lfloor 0.01\%N \rfloor$.

B. Related Work

As far as we know there is no algorithm solving *s-BLFE*, but *BFSS* is based on *Space Saving* which solves *ϵ-Deficient Frequent Elements*, besides, *s-BLFE* and *ϵ-Deficient Frequent Elements* can be seen as the two sides of a coin in some sense, so we present the representative algorithms solving *ϵ-Deficient Frequent Elements* here. Researches 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.

The 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.

Space Saving , the algorithm 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 algorithm *Count-Min Sketch* 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 algorithm *hCount* was proposed in [27]. The data structure and algorithms used in *Count-Min Sketch* and *hCount* are similar, but their focuses are different.

C. The challenges of s-BLFE

There are two main challenges of *s-BLFE* due to the different features between frequent items and low-frequency ones in data streams.

Long tail: There are at most $\lceil 1/s \rceil$ frequent items whose frequency is more than $\lfloor sN \rfloor$ in any data stream while 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 number of low-frequency items are much larger than that of frequent ones, and it's impossible to maintain them in memory. Another interesting observation is their frequencies are very low and close as well, and it will consume much memory space to separate low-frequency items from other items especially when s is very small.

Unpredictability: The basic idea of counter-based techniques is to discard potential infrequent items dynamically, which is based on the fact that potential infrequent items will never turn into frequent ones if they don't appear anymore, however, this no longer applies to low-frequency items because frequent items will possibly become low-frequency ones if they don't appear afterwards, which makes it difficult to find low-frequency items directly.

III. *BFSS* AND *BFSS**

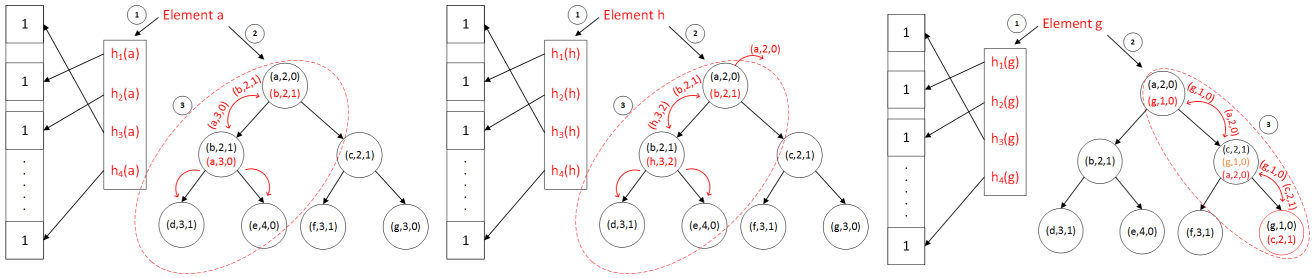
In this section, we present and discuss *BFSS* and *BFSS** in detail.

A. The BFSS Algorithm

Considering the challenges of *s-BLFE*, we solve the problem indirectly by filtering unsatisfactory items out.

Algorithm 1 and Algorithm 2 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 m counters, the form of which is $(e, f(e), \Delta(e))$. Each hash function maps an item from A to $[0, ..., K-1]$. Initially each bit of *BF* is set to 0 and all counters in D are empty. Fig. 2 shows the detailed execution process. We use a min heap to implement D. For each item e in S, we first map it to H bits and set the H bits to 1, then we check whether the element has been monitored in D, and the third step goes as follows: If the element is monitored, we increment the corresponding counter and then readjust the heap from its position to the bottom as Fig. 2a shows; If the element is not monitored and there is no available counter in D, we replace the item on the top of the heap which has the least hits, min, and change its counter to $(e, min+1, min)$, then we readjust the heap from the top to the bottom; If there is an available counter in D, as Fig. 2c shows, we assign a new counter $(e, 1, 0)$ to the item and insert the counter into the bottom of the heap and then readjust the heap from the bottom to the top.

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 not, it must not be a low-frequency item. If the item is in BF but not monitored D, we output it. If the item is in BF and monitored in 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.



(a) the element is monitored (b) D is full & the element is not monitored (c) D is not full & the element is not monitored

Fig. 2: The detailed execution process of *BFSS*

Algorithm 1 BFSS Update Algorithm

Input: Stream S, error parameter ϵ

$N = 0, m = 0, C = \lceil 1/\epsilon \rceil$;
for $i = 0$ to $K - 1$ **do**
  $BF[i] = 0$;
end for
for each item e in 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: support parameter s
Output: low-frequency items whose frequency is no more than $\lfloor sN \rfloor$

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

B. Analysis of BFSS

In this section, we present the theoretical analysis of *BFSS*, including *FNs*, *FPs*, space complexity, and time complexity. At last, we will identify the challenges to *BFSS*.

*1) **Analysis of FNs**:* In this section, we will prove there are no *FNs* in the output of *BFSS*. The proof is based on Lemma 1 to Lemma 5.

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

Proof: Every hit in S increments only one counter when D has available counters, and it is true even when a replacement happens because we add $f(e_m)$ to $f(e)$ and then increment $f(e)$. 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:

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

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 jth 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 jth 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:

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

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 not be low-frequency 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. ■

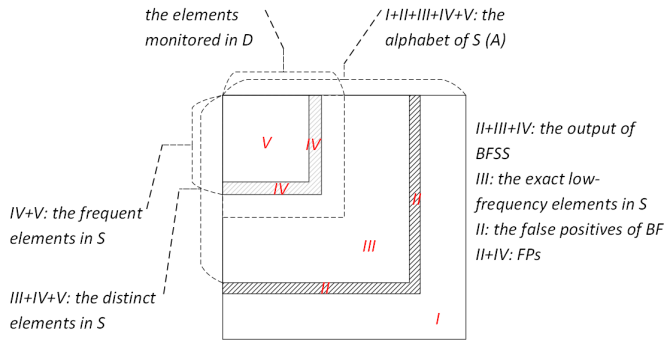


Fig. 3: The schematic diagram of the analysis of *BFSS*

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. 3 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:

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

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 α and $f_S(e_m)$, where $\alpha(> 0)$ and e_m denotes the item with the lowest frequency, we have:

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

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

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 α 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:

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

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:

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

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

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

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

$$H = \frac{K}{M} \times \ln 2 \quad (14)$$

Combined with Equation 11 and Equation 14, we have:

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

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.

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

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

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

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 the value of $g(n)$ decreases as the value of n increases, therefore we can calculate the value of n which makes the value of $f(n)$ maximum by solving the equation $g(n) = 0$, and the value of $F(n)$ increases firstly and then decreases.

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

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 $s = 0.1\%$. Fig. 4 shows the value of $F(n)$ when $M = 1M, K = 16M$ and $H = 11$ from which we can observe that the value of $F(n)$ increases firstly and decreases at last just as we have theoretically proved before, and when $n < 400000$, the value of $F(n)$ is nearly 0. Fig. 5 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 .5a shows that the value of n increases as the value of $\frac{K}{M}$ increases. From Fig. 5b, 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 has been discussed 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 $O(\lambda M + min(\lceil \frac{1}{\epsilon} \rceil, M))$. ■

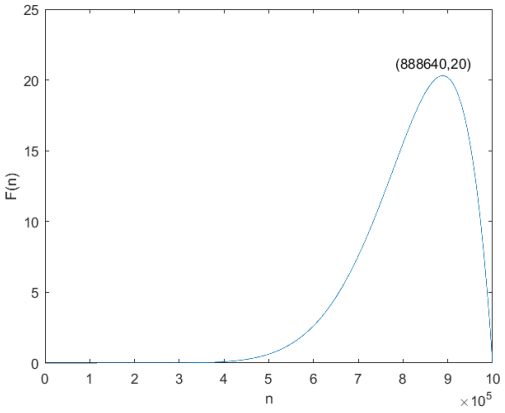
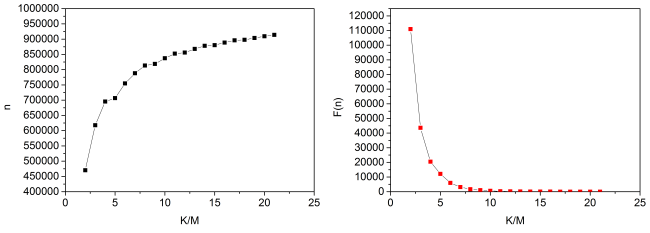


Fig. 4: The value of $F(n)$ when $M = 1M, K = 16M$ and $H = 11$



(a) (b)

Fig. 5: The maximum value of $F(n)$ and the corresponding value of n when $M = 1M$ - Varying $\frac{K}{M}$.

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 λ 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. ■

C. 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.

D. 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.

*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.

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 ⌊sN⌋, f'_s: the estimated total frequencies of the items whose frequency is above ⌊sN⌋, p'_1: the estimated percentage of the items whose frequency is above ⌊sN⌋, p'_2: the estimated percentage of the total frequencies of these items.}
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
        n' = n' + 1;
        if A[i] is monitored in D then
            if f(A[i]) > ⌊sN⌋ + Δ(A[i]) then
                n'_s = n'_s + 1;
                f'_s = f'_s + (f(A[i]) − Δ(A[i]));
            end if
        end if
    end if
end for
p'_1 = n'_s/n', p'_2 = f'_s/N;
Output p'_1, p'_2;
```

*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 $\Delta(p_2) =$

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

$p_2 - p_2'$, 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.

IV. Experiments

To evaluate the capabilities of *BFSS*, we conducted a comprehensive set of experiments, using both real and synthetic data. We compared the results against *nCount* and *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, lastly we compared F_1 as we are concerned with both precision and recall. We also measured the space used by each algorithm, which is essential to handle data streams. Notice that we included the size of the hash tables used in the algorithms for fair comparisons. Lastly, we summarize the experimental results. However, due to the page limit, we can not show the results of synthetic data here. 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], denoted as W, which consists of all the requests made to the 1998 World Cup Web site between April 30, 1998 and July 26, 1998. We extracted 7×10^7 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 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 the item is monitored, we simply increment its counter, otherwise if there exists an empty counter, we just insert a new counter into the heap, or 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.

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

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 $F(n)$.

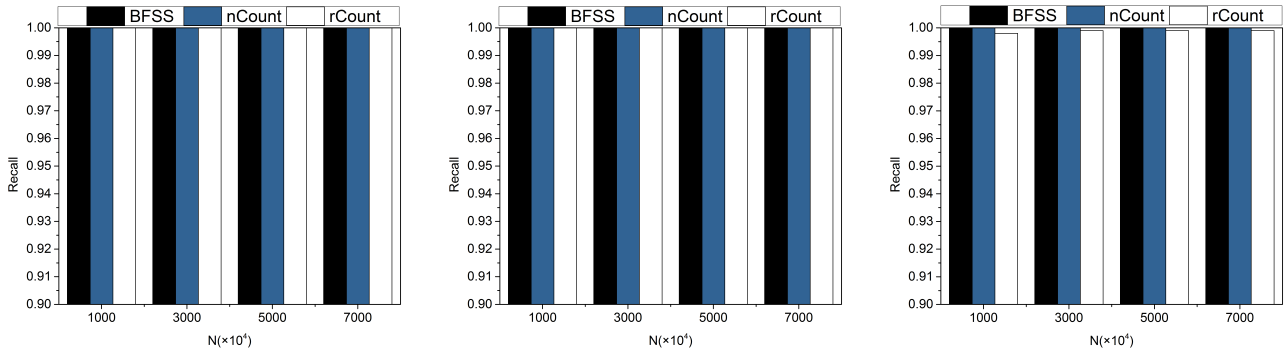
***nCount* Implementation.** The basic idea of *nCount* is to allocate each distinct item in S a counter, we use a hash table to index these counters to speedup updating, and it is time consuming if we use a linked list or an array. When an item comes, we increment the corresponding counter using hash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

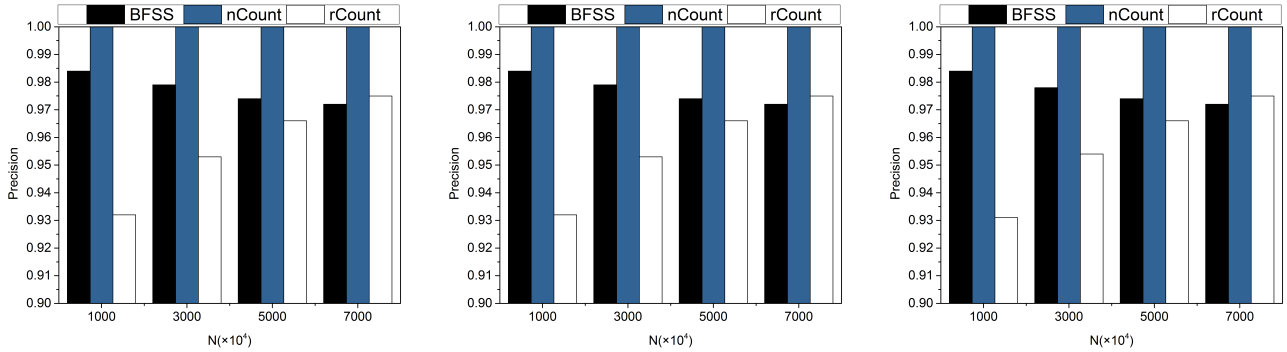
C. Performance of BFSS

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

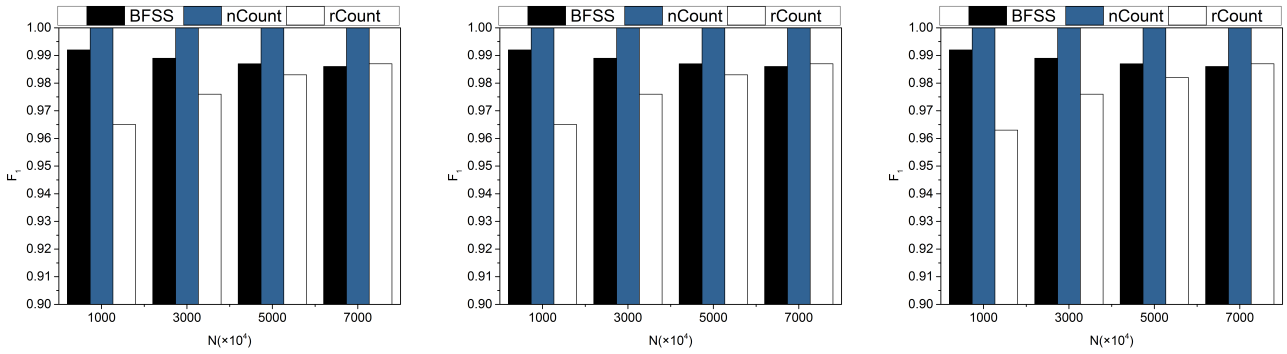
The results comparing the recall, precision, F_1 and space used by *BFSS*, *nCount* and *rCount* are summarized in Fig. 6. The value of N was varied from 10^7 to 7×10^7, and we set $K = 400000, H = 2$, in addition, we set the error parameter $\epsilon = s$, since our results showed that this was sufficient to give high accuracy in practice. 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. 6c. From Fig. 6d, Fig. 6e and Fig, 6f, 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* presented ealier decrease and later increase trend as the value of N increased with high probability. Fig. 7 shows the value of $F(n)$ when $M = 89753$ and $H = 2$ varying the value of K from which we can observe that the value of $F(n)$ increases firstly and reduces at last, when $K = 400000$, the maximum value of $F(n)$ is 2014 and the corresponding value of n is 56866, therefore the maximum value of $E(\#FPs)$ is $2014 + \frac{1}{s}$. The precision of *rCount* increased as the value of N increased because only the items not appearing in S may become false positives, and



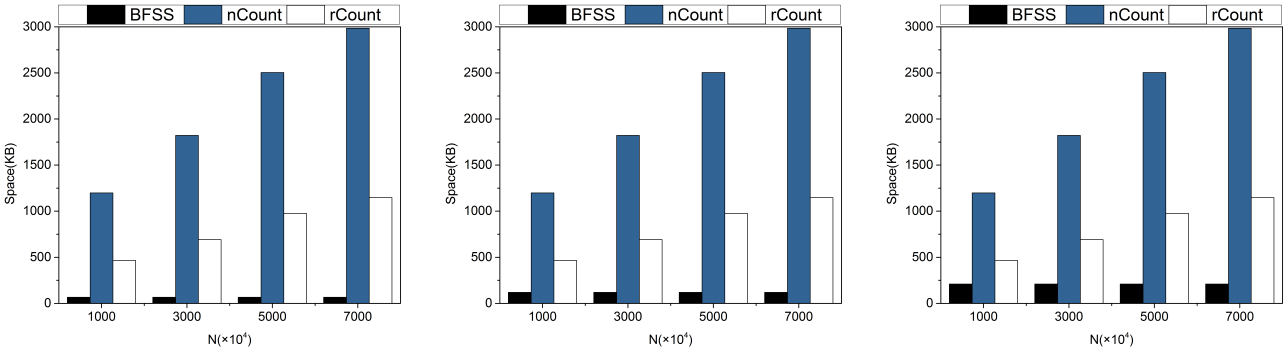
(a) Recall for 1%-BLFE (b) Recall for 0.2%-BLFE (c) Recall for 0.1%-BLFE



(d) Precision for 1%-BLFE (e) Precision for 0.2%-BLFE (f) Precision for 0.1%-BLFE



(g) F_1 for 1%-BLFE (h) F_1 for 0.2%-BLFE (i) F_1 for 0.1%-BLFE



(j) Space for 1%-BLFE (k) Space for 0.2%-BLFE (l) Space for 0.1%-BLFE

Fig. 6: Performance Comparison between *BFSS*, *nCount* and *rCount* on W - Varying s, ϵ and N

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

s	ϵ	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

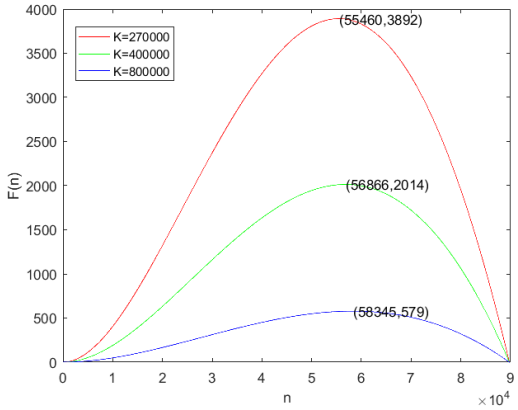


Fig. 7: The value of $F(n)$ when $M = 89753$ and $H = 2$ - Varying K

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. 6g, Fig. 6h and Fig. 6i show, and *BFSS* outperformed *rCount* except when $N = 7 \times 10^7$. The advantage of *BFSS* is evident in Fig. 6j, Fig. 6k and Fig. 6l, 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, in fact, from Fig. 7, we notice that the value of $F(n)$ decreases as the value of K increases, therefore the value of $E(\#FPs)$ decreases as the value of K increases too. Besides, once the values of K and N were confirmed the precision was almost confirmed too, which means the values of s and ϵ had little affection on the precison and most of *FPs* were caused by *BF*.

From above, *BFSS* achieved high precision and one hundred percent recall using much less space compared with *nCount* and *rCount*, in fact, we can give the maximum expected value of $E(\#FPs)$ theoretically. Besides, the space consumed by *BFSS* was stable as the value of N increased while the space consumed by *nCount* and *rCount* increased rapidly, which makes *BFSS* more scalable than *nCount* and *rCount*.

D. Distribution estimation

Table V shows the estimation of the distribution of the real data when N was 7×10^7, 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 V, 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, about thirty-two percent of the items in W were browsed only once, which is realy a "long tail".

TABLE V: Distribution Estimation of W.

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%

V. Conclusion

Low-frequency items in data streams are of great significance in areas like SEO because the total frequencies of them is even larger than that of frequent ones sometimes, and finding them is the first step to take good advantage of them, however, to the best of our knowledge, there is no algorithm which can effectively find them. We are the first to formally define the problem, named *s-BLFE*, and we present *BFSS* which can output all low-frequency items along with a few *FPs* using limited space, and the expected number of *FPs* can be theoretically bounded. The experimental results show *BFSS* achieve much space reductions with a little loss in precision compared with *nCount*.

The future directions of our work can be, but are not limited to: 1) output low-frequency items along with their approximate frequencies, which can be achieved based on the algorithm *Count-Min Sketch*. 2) support sliding window queries. 3) support both insertion and deletion of items. 4) find low-frequency item sets. In fact, many work done towards frequent items can be considered whether they are meaningful towards low-frequency ones, besides, the two main challenges of the problem *s-BLFE* mentioned before make it much more difficult to do further analysis on low-frequency items, which means we have a long way to go.

Acknowledgment

The authors would like to thank Haisu Zhang for his helpful suggestions.

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.