A Formal Analysis of Conservative Update Based Approximate Counting

Gil Einziger Roy Friedman
Computer Science Department
Technion
Haifa 32000, Israel
{gilga,roy}@cs.technion.ac.il

***Abstract*—This paper presents a formal analysis of multiple popular approximate counting schemes that employ the *conservative updates* policy, such as CU-sketch and Minimal Increment Spectral Bloom Filters, under a unified framework. It is also shown that when applied to items picked from a skewed distribution, such as Zipf-like functions, the analysis follows very closely empirical results obtained through simulations. Similar results are obtained when examining real Internet traffic traces and Wikipedia access traces. Furthermore, this paper's analysis is orders of magnitude more accurate than previously known analysis of approximate counting schemes.**

I. Introduction

The ability to efficiently count the frequency of items occurrences in a given domain is an important enabling tool for network monitoring and management, as well as several other domains. For example, networking applications utilize it for identifying high-load streams (also known as *heavy hitters*) and determining whether a given packet belongs to such a stream. In natural language processing applications, such schemes are used to hold the corpus of words and identify frequently appearing words. Caches can utilize approximate counting to determine the frequency of items in order to make educated cache replacement decisions. In social networks, this ability facilitates identifying popular users (sometimes called *social hubs*), etc.

What is common to all these applications is that they only require approximate counting, which can order items based on their frequency, but not necessarily give the exact number of occurrences. Moreover, due to the vast amounts of items they need to follow and their on-line decision making requirements, these applications require such methods to be both space and time efficient.

Several such methods, most of which are generalizations of *Counting Bloom Filters* and similar multiple hash-function based *sketches*, have been developed. Examples include, e.g., *Count Min Sketch* (CM-Sketch) [6], *Multi Stage Filters* [11] and *Spectral Bloom Filters* (SBF) [5]. These schemes share a similar structure and operation. Our work deals with an optimization suggested for these schemes called *Conservative Update* in [11] and *Minimal Increment* in [5]. It was introduced as a way to improve the accuracy of these schemes for applications that only perform positive increments (with no decrement).

It is common to believe that conservative update provides much higher accuracy for the same space. Yet, while it is known how to analyze the space vs. accuracy tradeoff of CM-Sketch, until now a similar analysis for conservative update schemes was not known. Instead, the use of these schemes relied on empirical tuning and educated guesses.

To highlight the benefits of conservative updates, we list a few well known examples of applications benefitting from its improved accuracy. In [11], the authors experimentally measured an accuracy improvement of two orders of magnitude on networking workloads. In [2, 22], conservative updates were used in order to monitor network traffic and discover heavy hitters. The work of [19] uses conservative updates in order to detect recently active heavy hitters. In [8], conservative updates are used in order to approximate the LFU cache admission policy. Further, [9] manages a distributed cache with conservative updates.

Conservative updates are also useful in other fields such as database systems [1, 23, 24], and natural language processing (NLP) [13, 14]. In these cases the amount of space required by an accurate data structure is prohibitory high and therefore an approximate solution is created. Conservative updates are then used to make the solution more space efficient.

In this paper, we study the conservative update variants of CM-Sketch and SBF under a unified framework and introduce a novel generalized construction called the *Layered Counting Sketch*. We use this construction to provide a formal analysis of such conservative update approaches. We validate the analysis through simulations and show that when the distribution is Zipf-like and known, our analysis follows very closely the simulation results and is order of magnitudes better than the previously known CM-Sketch analysis (which is oblivious to the distribution). We also show similar results when applying the analysis to real Internet TCP packets traces and Wikipedia access traces.

Let us also note that existing analysis of the CM-Sketch has the drawback that the approximation guaranteed δ depends solely on the number of hash functions used. Hence, another contribution of our analysis is in showing that well understood Bloom filter theory can be used in order to construct highly accurate conservative update sketches with only a small set of hash functions.

The rest of this paper is structured as follows: We discuss related work in Section II. We present preliminary definitions and assumptions in Section III while the analysis itself is introduced in Section IV. The simulation and trace driven results are shown in Section V. Finally, we conclude with a discussion in Section VI.

II. Related Work

As mentioned in the Introduction, approximated counting schemes are often used to maintain statistics on high speed network flows. This problem is particularly difficult because the update process has to be very quick. Network devices employ a hybrid DRAM/SRAM architectures to meet their timing constraints at an affordable hardware cost [21, 26, 29]. In these architectures, usually the lower bits of counters are stored in SRAM and occasionally the counters are dropped to DRAM. The challenges in this design choice is to make sure that not all the counters need to be flushed to DRAM at the same time, and that the communication between SRAM/DRAM is minimal.

Other methods are based on sampling [4, 10, 18], thereby avoiding performing an increment on every packet arrival. That is, counters are updated only for sampled packets. These methods indeed offer a smaller memory footprint, which allows them to store their data in faster SRAM.

CounterBraids uses a smart encoding and grouping of counters, which enables it to maintain its data structures entirely in SRAM, thereby obtaining fast updates [20]. However, CounterBraids suffers from very slow decode time. Alternatively, [17, 27] tackle the space overhead problem of a single counter by introducing compressed counters that requires fewer bits per counter but introduce an additional estimation error. These methods are complementary to the sketches discussed in this work, and can easily be deployed together.

Recently, [12] analyzed conservative updates and managed to analyze the uniform case using a technique called fluid approximation that allows them to model the counter growth rates using differential equations. Our analysis is much simpler, and can be applied to other distributions as well, and in particular to heavy-tailed ones that are considered more representative than uniform distributions.

III. Preliminaries

A. Bloom Filters

Bloom filters [3] are space efficient approximated data structures for answering set membership queries. Bloom filters support two methods: ADD and CONTAIN. A Bloom filter uses k hash functions, $h_1, h_2, ..., h_k$ to hash elements over an array of m bits. When adding an element T, $h_1(T), h_2(T), ..., h_k(T)$ are calculated and the matching bits of the array are set to 1.

The CONTAIN method hashes the item and tests the appropriate bits; if all the bits are set, the CONTAIN method returns true. If the Bloom filter is properly configured, this answer is usually correct. Yet, if one of the bits is unset, the item is guaranteed not to be contained in the set. We call the case when the CONTAIN method inaccurately includes an element in the set a *false positive*.

The false positive of a Bloom Filter that contains N elements is known to be $\left(1-\left(1-\frac{1}{m}\right)^{kN}\right)^k \approx \left(1-e^{-kN/m}\right)^k$. Since configuring Bloom filters is well understood, for our needs each Bloom filter configuration defines a false positive function $FP(X)$, where X is the number of items. Since our analysis uses the false positive function as a black box, a slightly different function can be used to adopt our analysis to a CM-Sketch with conservative updates (CU-Sketch) as explained in Section III-D.

B. Spectral Bloom Filters

Spectral Bloom Filters (SBF) [5] are an extension of Bloom filters in order to represent a multi set of items. An SBF can be seen as a Bloom filter with counters instead of bits. SBF supports two operations, ADD and ESTIMATE. In the ESTIMATE operation the item is hashed and all the associated counters are evaluated.

The minimal value of the counters is returned as the multiplicity estimation. In the ADD operation, the item is hashed and all the associated counters are evaluated and incremented.

Yet, as the ESTIMATE method only depends on the minimal value of the associated counters, we can increment only the counters whose value is the minimal value. This suggestion was called *Minimal Increment* in [5], yet it is the same as what is better known in the literature as *Conservative Update*. We refer to an SBF that utilizes the minimal increment technique an MI-SBF.

C. Count Min Sketch

As mentioned above, Count Min Sketch (CM-Sketch) [6] is an efficient and widely utilized construction for representing a multi set. Its interface contains two methods: ADD and ESTIMATE. This sketch is actually a two dimensional array of counters, the X axis is of size W and the Y axis is of size d. In CM-Sketch, we use d pairwise independent hash functions, each of domain $0, .., W$, and each one is responsible for one line in the two dimensional array.

The ADD and ESTIMATE operations of the CM-Sketch are identical to the operations of the SBF. The difference between them is mainly the type of hash functions they employ. A CM-Sketch limits the domain of each hash function to its own line, while in an SBF the domain of every hash function includes all the counters. Also CM-Sketch does not require the hash functions to be fully independent. Instead, it only requires pairwise independence. We refer to a CM-Sketch that employs the conservative update add operation as *CU-Sketch*. Unlike SBF, the approximation error of CM-Sketch has been formally analyzed. We note below the bottom line of the CM-Sketch analysis.

Definition Denote T_R the real value of element T, T_A the approximated value of T, and N the number of additions made to the sketch.

If we set $W = \frac{e}{\varepsilon}, d = \frac{1}{\ln(\delta)}$, we get the following guarantee: $\Pr\left(|T_A - T_R| < \varepsilon N\right) \geq 1 - \delta$. This means that with probability of $(1 - \delta)$ the estimated value of each element is within εN from its real value.

The count min sketch is illustrated in Figure 1. In this example, the multiplicity of the item that hashes to the marked counters is two. Adding the item again will increment all the hashed counters in the CM-Sketch and only the counter in the third (bottom) line in the CU-Sketch. Our analysis is also valid to the CU-Sketch, as explained in Section III-D below.

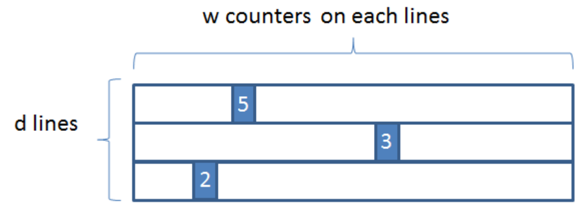


Fig. 1. A count min sketch illustration.

D. Generalized Bloom Filter Configuration

In order to apply the analysis to the CU-Sketch, we are required to define for each CU-Sketch configuration a false positive function. Consider a CU-Sketch with small 1 bit counters that satisfies the same interface as a Bloom filter. In particular, we say that the sketch contains an element if $h_1(T) = h_2(T) =, ..., h_d(T) = 1$. Hence, when adding an element, we only set the appropriate bits to 1. We call this case of a CU-Sketch a *Generalized Bloom Filter*.

We analyze the false positive probability in generalized Bloom filters in a similar way to Bloom filters. For simplicity, our analysis assumes that the d hash functions are independent.

The probability that an index is not set to one at a certain line after one addition is: $1 - \frac{1}{W}$. The probability that an index is not set to one at a certain line after N unique elements additions is: $\left(1 - \frac{1}{W}\right)^N$.

However, we are more interested in the probability that an index at a certain line is set to one after N unique additions. This probability is the inverse probability: $1 - \left(1 - \frac{1}{W}\right)^N$.

In order to experience a false positive, we require d indexes, from d different lines, to be set to one after N unique additions. Assuming the hash functions are independent, the probability for that is: $\left(1 - \left(1 - \frac{1}{W}\right)^N\right)^d$.

We conclude that the false positive probability is the same as the probability that d independent indexes are set to one after N insertions. Given a configuration (W, d), this function depends only on the number of unique elements that are added. In our work, the difference between the MI-SBF and the CU-Sketch is manifested in the choice of this false positive function. In particular, in an MI-SBF this function is the one provided by Bloom filter theory and in the CU-Sketch it is the one presented here.

IV. Analyzing Conservative Update Sketches

Below, we explore the connection between regular Bloom filters and MI-SBF/CU-Sketches. In particular, we introduce a new theoretical data structure named the *Layered Counting Sketch (LCS)* and explain how it is related to these popular constructions. The reason for working with this sketch is that its structure makes it easier to analyze than working directly with MI-SBF and CU-Sketch.

A. The Layered Counting Sketch

The layered counting sketch (LCS) is an array of Bloom filters. At each index of the array we have a Bloom filter. We call these indexes *levels*. Loosely speaking, we would like low level Bloom filters to contain both items of low and high frequency and high levels to contain only items of high frequency.

More accurately, when adding a new item to the LCS, we go over the levels from the first level up and add the item to the first level that did not already contain it. The second operation supported by the LCS is multiplicity estimation, which estimates how many times an item arrived in the past. To do that, we go over the levels of the LCS and return the index of the highest level that contains the item. A pseudo code of the LCS operations can be found in Algorithm 1.

An important observation is that the layered counting sketch actually count in unary base. Identical result can be obtained if we only remember what is the highest set level for each index. In this case we get an MI-SBF or a CU-sketch depending on the configuration. Therefore, any proof to the accuracy of the layered counting sketch also applies to MI-SBF/CU-Sketch. An LCS and the equivalent MI-SBF are illustrated in Figure 2. In that figure, we explain the addition process of an item, whose corresponding counters are highlighted. In the MI-SBF, we read counter values of 3,1, and increment the 1 counter to 2 (the number in red). Similarly, in the LCS, the added item is contained in the first level but is not contained in the second level and is therefore added to the second level (the red dot marks the changed bit).

Algorithm 1 Layered Counting Sketch (LCS) operations

```
function ADD(element T)
    for i = 0; i < TopLevel; i++ do
        if Not(Filter[i].contains(T)) then
            Filter[i].add(T))
            break
end function
function ESTIMATE(element T)
    for i = 0; i < TopLevel; i++ do
        if Not(Filter[i].contains(T)) then
            break
    return i
end function
```

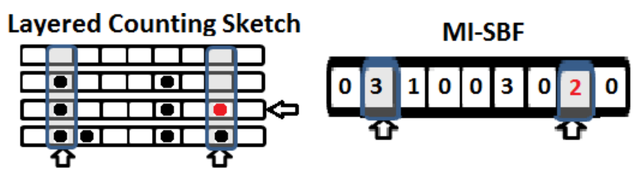


Fig. 2. Pseudo code of the LCS operations and an illustration of an LCS and the equivalent MI-SBF. Notice how the number of 1's in each LCS index, always equal to the value of the counter in the MI-SBF.

Definition Let T be an item and B be a Bloom filter. We denote $B[T]$ the set of indices associated with item T; hashing T using B's hash functions yields $B[T]$.

Lemma 4.1: After k insertions of item T to the LCS, $B[T]$ is set for all levels lower or equal to k.

Proof From Algorithm 1, when adding an item, we always insert an item to all the levels up until the first level that did not already contain it. Since T was inserted k times, we stopped at k different levels during k different insertions; each insertion is done by setting $B[T]$ at the appropriate level. Hence, in the first k levels $B[T]$ are set.

Lemma 4.2: **Containment lemma:** If the Bloom filter at level i has an index n set, then for every Bloom filter at level $i' < i$, index n is set.

Proof The proof follows from the construction: Index n was set by some insertion of some item T to the Bloom filter at level i. At the time of that insertion, $B[T]$ was set for all levels lower than i. $n \in B[T]$ and therefore n is set at the time of insertions. Also, our algorithm never unsets a bit and therefore n remains set for all levels lower than i.

Definition Denote by FP_i the false positive rate of level i.

Lemma 4.3: **Reduced error per level lemma:** If $i' < i$

then $FP_i \leq FP_{i'}$

Proof Consider two levels ($i' < i$) and an item (T) that was not inserted to any of the levels. By definition, the probability of a false positive for T at level i is FP_i. In that case, all indexes associated with T ($B[T]$) are all set at level i. Hence, the containment lemma guarantees that $B[T]$ are also set for level i'. We therefore have a false positive at level i'. The probability of such false positive is by definition $FP_{i'}$ and therefore $FP_i \leq FP_{i'}$.

1) Number of Items Per Level:

Definition For every i, denote by D_i the expected number of items that appear at least i times in the data. In our analysis, D_i is known and accurate.

Definition Denote A_i the actual number of unique items inserted to level i of an LCS.

An immediate observation from our model is that we already have a simple lower bound on the number of elements that are inserted per level. That is, because an element that appeared i times is inserted to at least i different levels, we conclude the following corollary:

Corollary 4.4: $\forall i$, $A[i] \geq D[i]$.

We now try to achieve an over estimation of $A[i]$, such an estimation will be translated to an upper bound.

B. Past False Positive Rate of Bloom Filters

From this point on, we assume for the purpose of the analysis that the distribution is known and i.i.d.. We leverage our knowledge about the distribution in order to construct a representing histogram of N items. In this histogram we aim to understand how many elements are expected to appear at each frequency for a population of N. We assume that N is very large, and therefore the standard deviation is negligible.

Notice that the false positive rate of a given Bloom filter corresponds to a specific configuration including a specific number of items that were already inserted into the Bloom filter. In particular, during an iterative process in which items are inserted one after the other, the false positive rate increases with each such insertion. For our purpose, the false positive rate and in particular the way it evolves is important to analyze the way the levels of an LCS get filled and, in particular, the number of elements that "skipped" a level due to a false positive at that level.

To that end, we define the notion of an *expected past false positive* as the answer to this question: Given a Bloom filter B of known configuration and N unique elements. Assume we first test each element to see wether it is contained in the filter and then insert it regardless. How many elements are expected to false positive in the process.

Definition Denote by $FP(N)$ the false positive of a Bloom filter configuration as a function of the number of elements N inserted to it.

Definition The *past false positive rate* of a Bloom filter B, denoted $PFP(N)$, is the average false positive rate in the above experiment.

$$PFP\,(N) = \frac{\sum_{i=1}^{N} FP\,(i)}{N}$$

We note that this function can be calculated for any Bloom filter configuration. Since the false positive function is monotonically increasing, it can also be very simply over approximated if direct calculation is too complex. This function is a conservative estimator in the case that the order of items' appearance is i.i.d since for that case, the first appearance of very frequent items is likely to occur even earlier than the average.

In order to provide an upper bound, we need to provide an upper estimation of A_i. To do so, we first bound the probability for elements to false positive at any certain number of levels.

Definition Denote T_R the number of times an item T appeared in the measurement and denote T_A our estimation of T_R.

Lemma 4.5: If $T_R \geq 1$ then $P\left(|T_A - T_R| > k\right) < PFP\,(A_k)$

Proof Since the error is one sided, it is enough to bound the probability of experiencing a false positive in k different levels. Clearly, if the item experienced a false positive in k different levels, the highest level that experienced a false positive cannot be lower than k. Since the input is i.i.d, the probability for an item to false positive at level k is at most $PFP(A_k)$.

However, the above formula assumes that we already know A_k, and we do not. What we do know is that $D_1 = A_1$ since the first level never receives false positives from previous levels. We can use this knowledge to upper bound A_2 in the following way: $E(A_2) \leq D_2 + (D_1 - D_2) \cdot PFP(A_1)$. The above formula represents our understanding about the way items are inserted to the second level. To do so, an item is required either to appear twice or more in the measurement (D_2), or to appear exactly one time in the measurement, but experience one or more false positives. The phrase

$D_1 - D_2$ is by definition the amount of items whose true frequency is exactly 1 and $PFP(A_1)$ is the bound on the probability of experiencing a single false positive.

We can continue upper bounding the expectancy of A_i using the following recursive formula.

$$E\left(A_i\right) = D_i + \sum_{j=1}^{i-1} \left(D_j - D_{j+1}\right) \cdot PFP\left(A_{i-j}\right)$$

C. Approximation Accuracy

Lemma 4.6: $P\left(|T_A - T_R| > k\right) < FP\left(A_k\right)$

Proof Since the error is one sided, it is enough to bound the probability of experiencing a false positive in k different levels. Clearly, if the item experienced a false positive in k different levels, the highest level that experienced a false positive cannot be lower than k. Unlike the previous case, since we do not know if the item appeared in the past, we cannot use the past false positive estimation. We can therefore only bound the probability by $FP(A_k)$. Intuitively, this means that items that did not appear in the sample are treated as if they arrived last. It also implies that the worst case accuracy happens when evaluating items that did not appear in the sample.

We now rephrase the above lemma in the way accuracy of sketches is presented in the literature, i.e., we require an approximation guarantee. This guarantee assures us that for each item, the approximated value is close to the real value with high probability. We use two parameters to define it: ϵ indicates how close the approximated value and the real value are while δ indicates the amount of certainty in that claim. We therefore rephrase the above lemma to provide an approximation guarantee for the LCS.

Corollary 4.7: $P\left(|T_A - T_R| < \varepsilon N\right) \geq 1 - \delta$ for $\lceil \varepsilon N \rceil = k$ and $FP\left(A_{\lceil \varepsilon N \rceil}\right) = \delta$

Our upper estimation about A_i can now be translated to an appropriate upper bound. In a similar way, we can use our lower estimation about A_i to generate the lower bound:

$$P\left(|T_A - T_R| > \varepsilon N\right) \geq FP\left(D_{\lfloor \varepsilon N \rfloor}\right)$$

This result has the benefit that we can determine for a specific configuration the appropriate δ for each ϵ. Moreover, since the problem is reduced to Bloom filter theory, we can utilize our existing knowledge of Bloom filters to configure an LCS properly. In particular, we can keep δ arbitrary small even with a small number of hash functions, a property that may be appealing for practical applications. This is an improvement over the CM-Sketch analysis that requires additional hash functions in order to reduce δ.

V. Results

A. Theory vs. Simulations

Since our analysis can be applied to any known distribution, we chose to experimentally evaluate them for several representative Zipf-like distributions, to see how they compare with the actual results and to previously known analysis of CM-Sketch. To do so, we evaluated both our upper and lower bounds for a window of 10 million i.i.d requests. We implemented both the CU-Sketch and the MI-SBF and added 10 million items from the same distribution to it. The simulations are performed using our Java based prototype of LCS. For the known CM-Sketch analysis, we use the analysis given by the original authors and when applicable the improved analysis for skewed distributions in [7].

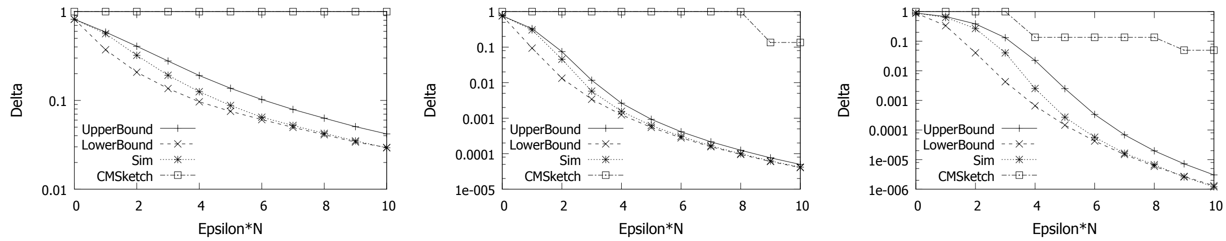
We focus on two types of MI-SBF configurations: *Accurate* configurations are created by picking the optimal number of hash functions and counters to provide minimal false positive at the first level. These configurations are usually accurate enough to apply the CM-Sketch analysis and get a meaningful result. Similar configurations were used by [8].

We also used *inaccurate* configurations. In these configurations 3 hash functions are always used and the false positive rate at the first level is close to 100%. However, since the majority of items in practice are low frequency items, they populate the low levels of these sketches while the high levels remain relatively empty. Similar configurations are useful for heavy hitters detection and security applications [25].

The evaluation of the approximation guarantee was done by examining the εN and δ values obtained for items that were not already included in the sketch. This is because such items experience the worst case false positive rate (see Lemma 4.3).

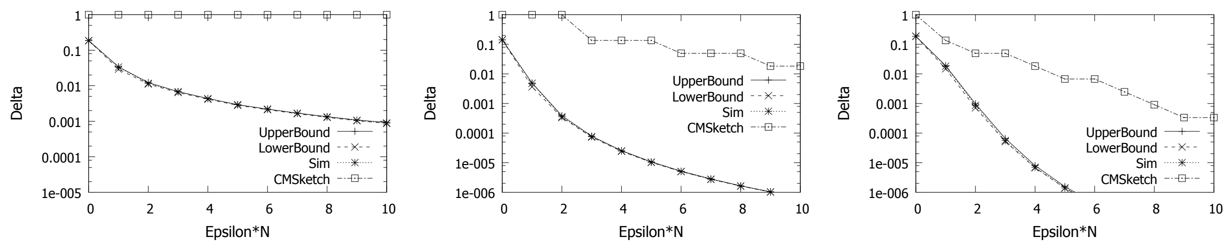
The results of the inaccurate configurations are shown in Figure 3. As can be observed, our analysis captures well the behavior of LCS. The upper bound here is not tight, yet it is close to the actual behavior of the sketch. Recall that no formal analysis of the CM-Sketch can be applied for these configurations and therefore our analysis is the only one that is applicable.

As for the accurate configurations, which can be reasoned about using existing CM-Sketch analysis, the results are depicted in Figure 4. As can be seen, our analysis is orders of magnitude more accurate than existing CM-Sketch analysis. It is also significantly more



(a) Very skewed Zipf 2 distribution (b) Skewed + heavy tailed Zipf 1 distribution (c) Lightly skewed Zipf 0.6 distribution

Fig. 3. The accuracy of the analysis vs. simulations and previous analysis (inaccurate configuration).



(a) Very skewed Zipf 2 distribution (b) Skewed + heavy tailed Zipf 1 distribution (c) Lightly skewed Zipf 0.6 distribution

Fig. 4. The accuracy of the analysis vs. simulations and previous analysis (accurate configuration). Notice that both the simulated result and the upper and lower bounds are almost identical!

accurate than the previous case. The reason for this is that in the accurate configurations, $A_i \approx D_i$ and therefore the analysis is very accurate.

B. Order of items

In the experiments above, the items appeared by independently picking random numbers from a given distribution. While our lower bound is indifferent to item ordering, our upper bound assumes that the arrival process is i.i.d. (but obviously not necessarily uniform). Yet, realistic networking workloads are often not i.i.d. In order to evaluate non-i.i.d workloads, we decided to order the items in two extreme orders: The first ordering is *Most Frequent First (MF First)*. In this workload, the most frequent items appear first. This workload is likely to minimize the error for large $Epsilon \cdot N$ values, since the probability of high frequency items to false positive in this workload is minimal.

Our second ordering, *Least Frequent First (LF First)*, is designed to maximize the error for large $Epsilon \cdot N$ values. To do so, we insert first the low frequency items, and only then the high frequency ones. As a result, when the high frequency items are inserted to the sketch, the false positive rate is at its pick. These items are likely to false positive more than the average item causing more items to be inserted to high levels.

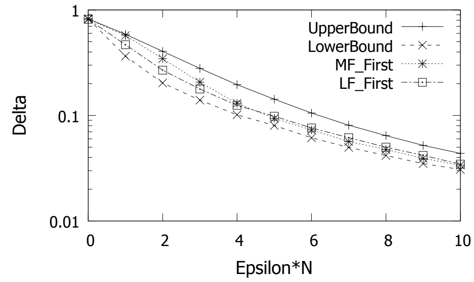
We stress that these workloads violate our assumptions, and therefore the upper bound does not formally apply to them. Still, as can be seen in Figure 5, our analysis is highly accurate even for these extreme orderings. Our lower bound, as expected, is indeed lower than any of the other curves. The upper bound is still very indicative to the performance of the system. The only cases where it does not hold are large values of $Epsilon \cdot N$ with the Zipf 1 workload, and even there the difference is insignificant.

Hence, we can conclude from the result of this experiment that our analysis yields very accurate indications even when considering non-i.i.d. workloads.

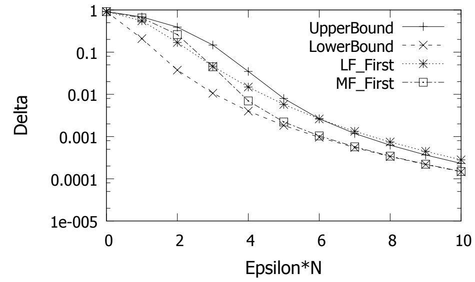
C. Evaluation over real life traces

Below, we evaluate the accuracy of our analysis using real life traces. We used [15] and [16] in order to count how many packets are transferred on each TCP flow. We also used a very extensive Wikipedia trace [28] in order to count how many times each resource is requested.

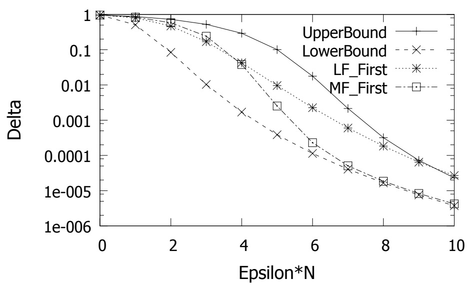
We note that our formal analysis is not valid in this case since the distribution is not constant and known and the traffic pattern is not i.i.d.. Yet, we applied it regardless, by taking a single 10 million items sample, and assumed that the distribution over this sample is constant. We used multiple different consecutive 10



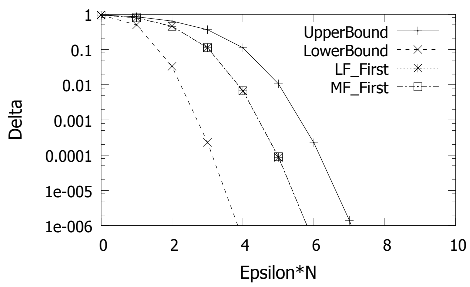
(a) Very skewed Zipf 2 distribution



(b) Skewed + heavy tailed Zipf 1 distribution



(c) Lightly skewed Zipf 0.6 distribution



(d) Uniform distribution

Fig. 5. Effects of the item ordering on the accuracy of the sketch

million slices from the traces in order to evaluate the simulated accuracy.

Our results are illustrated in Figure 6. As can be observed, our analysis remains very indicative for both per flow packet counting and for HTTP request counting. In the context of real workloads, the upper and lower bounds should be viewed us upper and lower estimators.

VI. Discussion

This paper includes an accuracy analysis of approximate counting schemes that utilize the conservative update approach. Such schemes have many applications in various domains, and in particular in network monitoring and management. To obtain the analysis, we have introduced a novel theoretical layered data structure called layered counting sketch (LCS). The benefit of LCS is that it can be reasoned directly using Bloom filters theory, which enabled us to obtain a rigorous yet simple analysis. As we have shown, known sketches such as CU-Sketch and MI-SBF can be mapped to LCS, which serves as a unified vehicle for their accuracy analysis. Our analysis conforms very well with simulations results using synthetic Zipf-like workloads and when applied to real life Internet packets traces and Wikipedia traces.

Our bounds can potentially save a significant amount of space when configuring sketches in an analytic way. For example, in order to approximately count 10 million items distributed according to Zipf 2, we can use a CU-sketch with 5,800 counters and 4 hash functions. According to our upper bound, this configuration yields $\delta = 1.2\%$ for $\varepsilon \cdot N = 20$.

In comparison, applying the improved CM-Sketch analysis of [7] for $\varepsilon \cdot N = 20$ indicates that such a CU-Sketch would require 6,300 counters per line. Further, 4 lines are needed to achieve $\delta < 2\%$ and consequently the produced sketch is more than 4 times larger than what our upper bound determines.

Looking into the future, we would like to learn how to ideally configure an MI-SBF/CU-Sketch for a minimal memory consumption given a specific δ and ε. We would also like to analyze the accuracy of these sketches when coupled with counter compression methods like [17, 27].

References

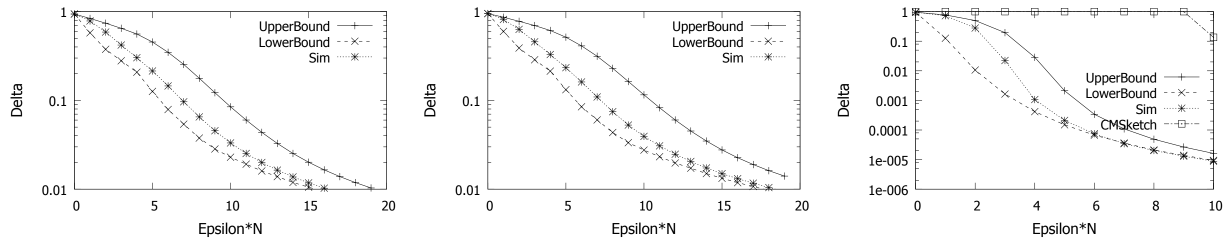
[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '99, pages 10–20, New York, NY, USA, 1999. ACM.

[2] G. Bianchi, S. Teofili, E. Boschi, B. Trammell, and C. Greco. Scalable and fast approximate excess rate detection. In *Future Network and Mobile Summit, 2010*, pages 1 –9, june 2010.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[4] B.-Y. Choi, J. Park, and Z.-L. Zhang. Adaptive random sampling for load change detection. *SIGMETRICS Perform. Eval. Rev.*, 30(1):272–273, June 2002.

[5] S. Cohen and Y. Matias. Spectral bloom filters. In *Proceedings of the 2003 ACM SIGMOD international conference on Man-*



(a) [15] – inaccurate configuration (b) [16] – inaccurate configuration (c) Wikipedia – inaccurate configuration

Fig. 6. Analysis accuracy with real world Internet packets traces and the Wikipedia trace.

agement of data, SIGMOD '03, pages 241–252, New York, NY, USA, 2003. ACM.

[6] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55:29–38, 2004.

[7] G. Cormode and S. Muthukrishnan. Summarizing and mining skewed data streams. In *Proc. of the 2005 SIAM International Conference on Data Mining*, pages 44–55, 2005.

[8] G. Einziger and R. Friedman. Tinylfu: A highly efficient cache admission policy. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 146–153, Feb 2014.

[9] G. Einziger, R. Friedman, and Y. Kantor. Shades: Expediting kademlias lookup process. In *Euro-Par 2014 Parallel Processing*, volume 8632 of *Lecture Notes in Computer Science*, pages 391–402. Springer International Publishing, 2014.

[10] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, Aug. 2004.

[11] C. Estan and G. Varghese. New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4):323–336, Aug. 2002.

[12] D. J. L. G. Bianchi, K. R. Duffy and V. Shneer. Modeling conservative updates in multi-hash approximate count sketches. In *ITC 24*, 2012.

[13] A. Goyal, H. Daumé, III, and G. Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1093–1103, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[14] A. Goyal and H. D. III. Lossy conservative update (lcu) sketch: Succinct approximate count storage. In W. Burgard and D. Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

[15] P. Hick. CAIDA Anonymized 2008 Internet Trace, equinix-chicago 2008-03-19 19:00-20:00 UTC, Direction A. http://www.caida.org/data/monitors/passive-equinix-chicago.xml.

[16] P. Hick. CAIDA Anonymized 2008 Internet Trace, equinix-chicago 2008-03-19 19:00-20:00 UTC, Direction B. http://www.caida.org/data/monitors/passive-equinix-chicago.xml.

[17] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, C. Wu, and Y. Cheng. Disco: Memory efficient and accurate flow statistics for network measurement. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 665–674, Washington, DC, USA, 2010. IEEE Computer Society.

[18] C. Hu, S. Wang, J. Tian, B. L. 0001, Y. Cheng, and Y. Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *INFOCOM*, pages 26–30. IEEE, 2008.

[19] P. Kolaczkowski. Memory efficient algorithm for mining recent frequent items in a stream. In *Proceedings of the international conference on Rough Sets and Intelligent Systems Paradigms*, RSEISP '07, pages 485–494, Berlin, Heidelberg, 2007. Springer-Verlag.

[20] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. Counter braids: a novel counter architecture for per-flow measurement. In *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '08, pages 121–132, New York, NY, USA, 2008. ACM.

[21] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counter architecture. *SIGMETRICS Perform. Eval. Rev.*, 31(1):261–271, June 2003.

[22] F. Raspall and S. Sallent. Adaptive shared-state sampling. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, IMC '08, pages 271–284, New York, NY, USA, 2008. ACM.

[23] F. Rusu and A. Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 187–198, New York, NY, USA, 2007. ACM.

[24] F. Rusu and A. Dobra. Sketches for size of join estimation. *ACM Trans. Database Syst.*, 33(3):15:1–15:46, Sept. 2008.

[25] O. Salem, S. Vaton, and A. Gravey. A scalable, efficient and informative approach for anomaly-based intrusion detection systems: Theory and practice. *Int. J. Netw. Manag.*, 20(5):271–293, Sept. 2010.

[26] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in router line cards. *IEEE Micro*, 22(1):76–81, Jan. 2002.

[27] E. Tsidon, I. Hanniel, and I. Keslassy. Estimators also need shared values to grow together. In A. G. Greenberg and K. Sohraby, editors, *INFOCOM*, pages 1889–1897. IEEE, 2012.

[28] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009.

[29] Q. Zhao, J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '06/Performance '06, pages 323–334, New York, NY, USA, 2006. ACM.