Finding (Recently) Frequent Items in Distributed Data Streams

Amit Manjhi Vladislav Shkapenyuk Kedar Dhamdhere
Christopher Olston

April 2004
CMU-CS-04-12U

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Keywords: Streams and Stream-based Processing, Optimization and Performance

Abstract

We consider the problem of maintaining frequency counts for items occurring frequently in the union of multiple distributed data streams. Naïve methods of combining approximate frequency counts from multiple nodes tend to result in excessively large data structures that are costly to transfer among nodes. To minimize communication requirements, the degree of precision maintained by each node while counting item frequencies must be managed carefully. We introduce the concept of a *precision gradient* for managing precision when nodes are arranged in a hierarchical communication structure. We then study the optimization problem of how to set the precision gradient so as to minimize communication, and provide optimal solutions that minimize worst-case communication load over all possible inputs. We then introduce a variant designed to perform well in practice, with input data that does not conform to worst-case characteristics. We verify the effectiveness of our approach empirically using real-world data, and show that our methods incur substantially less communication than naïve approaches while providing the same error guarantees on answers.

In addition, we extend techniques for maintaining frequency counts of high-frequency items in one or more streams by making them time-sensitive. Time-sensitivity is achieved by associating weights with items that decay exponentially with time. We analyze the error bounds and worst-case space bounds for the extended algorithms.

1 Introduction

The problem of identifying frequently occurring items in continuous data streams has attracted significant attention recently [4,8,10,13,17,20]. Potential applications include identifying large network flows [10], answering iceberg queries [11], computing iceberg cubes [15] and finding frequent itemsets and association rules [1].

However, earlier work on identifying frequent items in data streams and estimating their occurrence frequencies falls short of meeting the needs of the many real-world applications that exhibit one or both of the following two properties:

1. Distributed streams. Streams originate from multiple distributed sources. The data needs to be aggregated to arrive at the final result, as in the distributed streams model of [12]. This situation occurs, for example, if we wish to detect frequent events in a sensor network environment.

2. Time sensitivity. Recent data is more important than older data. For example, in telecommunications, most processing focuses on recent call records [7].

We briefly describe two real-world applications exhibiting the properties just mentioned:

Detecting DDoS Attacks. Early detection of *Distributed Denial of Service* (DDoS) attacks is an important topic in network security. While a DDoS attack typically targets a single "victim" node or organization, there is generally no common path that all packets take. In fact, even packets sent to the same destination and originating from within the same organization may follow different routes, due to so-called "hot potato" routing [3]. This property makes it very difficult to detect distributed denial of service attacks effectively by only considering the traffic passing through any single monitoring point, and motivates a distributed monitoring approach. Furthermore, techniques that weigh recent data more than past data may help in early detection of attacks.

Usage Monitoring in Large-scale Distributed Systems. Web content providers using the services of a *Content Delivery Network* (CDN) like Akamai [2] may wish to monitor recent access frequencies of content served (e.g., HTML pages/images), to keep tabs on current "hot spots." The CDN may serve requests from any of a number of cache nodes (Akamai currently has over 10,000 such nodes); typically requests are served by the cache node closest to the end-user making the request in order to minimize latency. Hence, keeping tabs on overall access frequencies requires distributed monitoring across many CDN cache nodes.

1.1 Problem Variants

Both applications outlined above require algorithms for identifying recent high-frequency items in the union of many distributed streams, and estimating the corresponding occurrence frequencies. In general, we can classify applications of frequent item counting into four categories, in terms of whether they require time-sensitivity and distributed monitoring capability, as shown in Table 1. We briefly describe each problem variant:

(1) Finding frequent items in a single stream: A single node sees an ordered stream of possibly repeating items. The goal is to maintain frequency counts of items whose frequency currently exceeds a user-supplied fraction of the size of the overall stream seen so far.

(2) Finding recently frequent items in a single stream: In this variant recent occurrences of items in the stream are considered more important than older occurrences of items. At any given time, a numeric weight is associated with each item occurrence in the stream that is a function of the amount of time that has elapsed since the appearance of the item in the stream. A commonly-used weighting scheme is *exponential decay* [6], in which weights are assigned according to a negative-exponential function of elapsed time. The goal is to identify items whose cumulative weighted frequency currently exceeds a user-supplied fraction of the total across all items, and provide an estimate of the cumulative weighted frequencies of any such items.

(3) Finding frequent items in the union of distributed streams: In this variant there are m ordered streams $S\backslash, 5*2, \bullet. \bullet, S_m$, each produced at a different node in a distributed environment and consisting of a sequence of item occurrences. The goal is the same as in Variant (1), except that item frequencies are computed over the union of streams $S'i, \$'2, \ldots, 5_m$, instead of over a single stream.

Table 1: Problem variants.

	Single Stream	Distributed Streams
Time-insensitive	(1)	(3)
Time-sensitive	(2)	(4)

(4) Finding recently frequent items in the union of distributed streams: This variant represents the natural combination of Variants (2) and (3).

Of these four variants, only Variant (1) has been studied in prior work. Algorithms for time-insensitive frequency counting over a single stream include those presented in [8, 17, 20]. While it is relatively straightforward to extend these algorithms to handle Variant (2), the effect on the space bounds and error guarantees of the resulting algorithms in some cases is nonobvious. In this paper we provide rigorous analysis of these aspects.

Variants (3) and (4) present a larger challenge. As we will show, simple adaptations of existing frequency counting algorithms to work in a distributed setting incur excessive communication. In this paper we present a new framework for distributed frequency counting that minimizes communication requirements. Before outlining our approach we first provide a formal problem statement that unifies the four variants listed above.

1.2 Unified Problem Statement

Our problem statement extends that of [20]. There are $m \geq 1$ ordered data streams $S_1, S_2, \ldots, S_m$. Each stream S_i consists of a sequence of item occurrences with time-stamps: $\langle o_{i1}, t_{i1}\rangle, \langle o_{i2}, t_{i2}\rangle$, etc. Each item occurrence o_{ij} is drawn from a fixed set U of items, i.e., $\forall i, j, o_{ij} \in U$. Arbitrary repetition of item occurrences in streams is allowed. Each stream S_i is monitored by a corresponding *monitor node* M_i, of which there are m. Monitored frequency counts for high frequency items are to be supplied to a central *root node* R, which may or may not be the same as one of the monitor nodes.

Let S be the sequence preserving union of streams $S_1, S_2, \ldots, S_m$. Further, let $c(u)$ be the frequency of occurrence of item u in S up to the current time, weighted by recency of occurrence in an exponentially decaying fashion. Mathematically,

$$c(u) = \sum_{\langle o_i, t_i\rangle \in S, o_i = u} \alpha^{\lfloor \frac{t_{now} - t_i}{T} \rfloor}$$

where t_{now} denotes the current time, and α and T are user-supplied parameters. The parameter $\alpha \in (0, 1]$ controls the aggressiveness of exponential weighting. As a special case, setting $\alpha = 1$ causes all item occurrences to be weighted equally, regardless of age (as in Variants (1) and (3) of Section 1.1). The parameter $T > 0$ controls the frequency with which answers are reported, and also the granularity of time-sensitivity. A time period of T time units is referred to as an *epoch*.

The objective is to supply, at the end of every epoch (i.e., every T time units), an estimate $\hat{c}(u)$ of $c(u)$ for items occurring in S whose true time-weighted frequency $c(u)$ exceeds a *support threshold* $\mathcal{T}$. $\mathcal{T}$ is defined as the product of a user-supplied *support parameter* $s \in [0, 1]$, and the sum of the weighted item occurrences seen so far on all input streams, $N = \Sigma_{u \in U} c(u)$, i.e., $\mathcal{T} = s \cdot N$. The amount of allowable inaccuracy in the frequency estimates $\hat{c}(u)$ is governed by a user-supplied parameter ϵ. It is required that $0 \leq \epsilon \leq s$ (usually, $\epsilon \ll s$). Each time an answer is produced, it must adhere to the following guarantees:

1. All items whose true time-weighted frequency exceeds $s \cdot N$ are output.
2. No item whose true time-weighted frequency is less than $(s - \epsilon) \cdot N$ is output.
3. Each estimate $\hat{c}(u)$ supplied in the answer satisfies: $\max\{0, c(u) - \epsilon \cdot N\} \leq \hat{c}(u) \leq c(u)$.

A useful data structure for storing intermediate answers is an (e, *a)-synopsis* of item frequencies over a stream or union of several streams. An (e, a)-synopsis *S* consists of a (possibly empty) set of time-weighted frequency estimates each denoted *S:ĉ(u),* where each *ĉ(u)* estimate satisfies max {0, *c(u) - c-S:n}* $\leq$ *ĉ(u)* $\leq$ *c(u). S:n* denotes the total time-weighted frequency of all items in the synopsis *(S:n = J2$_u$eu $^{c(w)}$)*- The salient property of an (e, a)-synopsis is that items with weighted frequency below *e-S:n* need not be stored, resulting in a reduced-size representation.

1.3 Overview of this Paper

Finding recently frequent items (Section 2): To begin, we show how to extend two recent frequency counting algorithms that produce (e, l)-synopses to produce (e, a)-synopses, for any *a e* (0,1], to achieve Variant 2 of Table 1. We analyze the correctness and space requirements of the resulting algorithms. In particular, we show that the worst-case size of time-sensitive synopses is bounded by a time-independent constant.

Finding (recently) frequent items in distributed streams (Section 3): There are two obvious, simple strategies for adapting single-stream frequency counting algorithms to a distributed setting to achieve Variants 3 and 4 of Table 1, and both have serious drawbacks:

SSI: Periodically, at the end of every epoch, each monitor node *Mi* sends to the root node *R* the exact frequency counts of all items occurring in *Si* over the last *T* time units. Node *R* then combines the counts received from the monitor nodes with (possibly time-decayed) counts maintained over prior epochs, and outputs items whose overall weighted counts exceed the support threshold T.

SS2: Each monitor node *Mi* maintains an (e, l)-synopsis *Si* over the recent portion of its local stream *Si.* Intuitively, the (e, l)-synopsis is a reduced summary of item frequencies that does not include items whose frequency in *Si* is small. Periodically, at the end of every epoch, each *Mi* sends its local synopsis *Si* to node *R.* Upon receiving all local synopses, node *R* combines them into a single unified (e, l)-synopsis containing estimated item frequencies for the union of the contents of all input streams in the most recent epoch. This synopsis is then combined additively with an (e, a)-synopsis containing estimated weighted counts from previous epochs, after multiplying those synopsis counts by a, to generate a new (e, a)-synopsis valid for the current epoch. Lastly, items whose estimated time-decayed counts exceed the support threshold T (after taking into account the error tolerance) in this synopsis are output[1].

Clearly, strategy SSI is likely to incur excessive communication because frequency counts for all items, including rare ones, must be transmitted over the network. Furthermore, the root node *R* must process a large number of incoming counts. While strategy SS2 alleviates load on the root node to some extent, in the presence of a large number of monitor nodes and rapid incoming streams, the root node may still represent a significant bottleneck. To further reduce the load on the root node, nodes can be arranged in a hierarchical communication structure (see Figure 1), in which synopses are combined additively at intermediate nodes as they make their way to the root. In this setting SS2 compresses data (by dropping small counts) as much as possible at each leaf node without violating the e error bound. Consequently no further compression can be performed as synopses are combined on their way to the root or at the root node itself, making it impossible to eliminate counts for items whose frequency exceeds e fraction of one or more individual streams but does not exceed e fraction in the union of the streams whose synopses are combined at a non-leaf node. Hence, if input streams have different distributions of item occurrences, counts for items of small frequency may reach the root node unnecessarily under strategy SS2.
There are thus two main disadvantages of using SS2:

1. High communication load on root node *R.*
2. High space requirement on *R.*

Suppose that, instead of applying maximal synopsis compression at the leaf nodes, some compression capability is reserved until synopses of multiple incoming streams are combined at non-leaf nodes. If that is done, more aggressive

[1] Note that in both strategies time-sensitivity is only introduced at node *R.* It is not possible to introduce time-sensitivity in data before it is sent to *R,* since all item frequencies in the most recent epoch have weight 1 in our formulation.

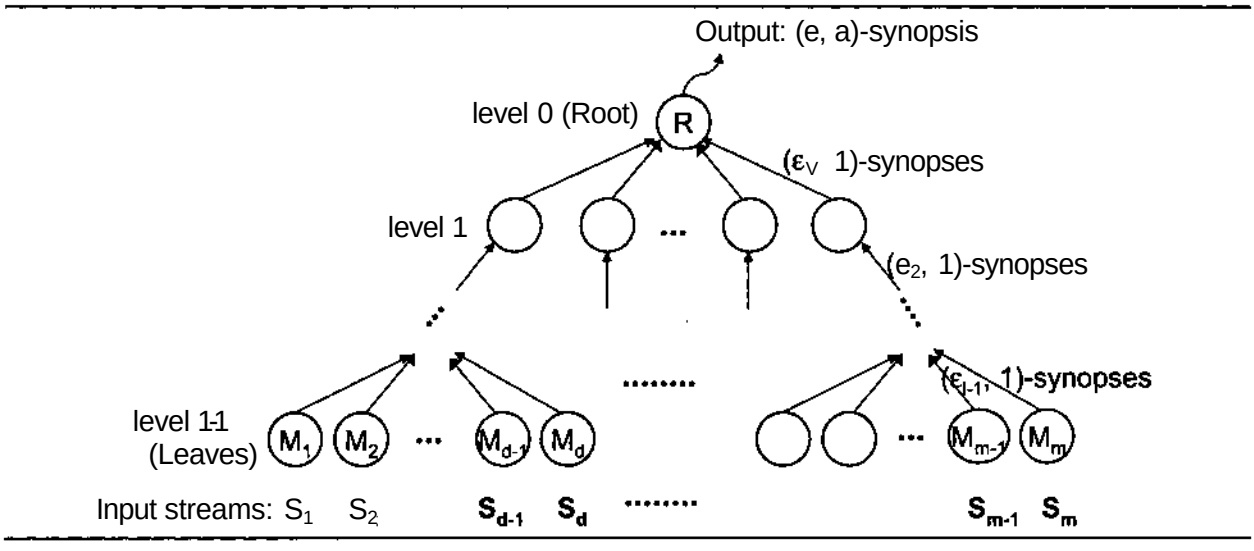


Figure 1: Hierarchical communication structure.

compression can be performed by non-leaf nodes by taking into account the distributions of item frequencies over a larger set of input streams. As a result, the synopses reaching the root (and the synopsis maintained over previous epochs at the root) will likely be significantly smaller than in SS2. On the other hand, the synopses passed from the leaf nodes to their parents may be larger than in SS2, which is an undesirable side-effect.

Indeed, to avoid excessive communication load on any particular node or link, the amount of compression performed by each node while creating or combining synopses must be managed carefully. In hierarchically-structured monitoring environments we can configure the amount of compression performed, and consequently, the amount of error introduced at each level so that synopses follow a *precision gradient* as they flow from leaves to the root. It turns out that worst-case communication load on any link is minimized by using a gradual precision gradient, rather than either deferring the introduction of error entirely until data reaches the root (as in SSI), or introducing the maximum allowable error at the leaf nodes (as in SS2). Still, the best gradual precision gradient to use is not obvious.

In this paper we study the problem of how best to set the precision gradient formally. We first show how use of a gradual precision gradient alleviates storage requirements at the root node *R.* Then, we derive optimal settings of the precision gradient under two objectives: (a) minimize load on the root node *R,* and (b) minimize maximum load on any single communication link under worst-case input behavior. We then introduce a variant that aims to achieve low load on all links in practice, when input data may not exhibit worst-case characteristics, by exploiting a small sample of the expected input data obtained in advance.

Remainder of paper: In Section 4 we confirm our analytical findings of Sections 2 and 3 through extensive experimental evaluation on three real-world data sets. Our experiments demonstrate that naive methods of finding frequent items in distributed streams (SSI and SS2) can incur high communication and storage costs compared with our methods. Related work is discussed in Section 5, and we summarize the paper in Section 6.

2 Finding Recently Frequent Items

Given a simple frequency counting algorithm that maintains a (0, l)-synopsis, i.e., one with exact, time-insensitive frequency counts for all items, it is straightforward to add time-sensitivity in the form of exponential weighting by some desired $a \in (0,1]$. By multiplying each count in the synopsis by a once every T time units, we achieve a (0, a)-synopsis. It is tempting to apply the same method to extend approximate frequency counting algorithms such as [8,17,20] that maintain (e, l)-synopses to instead maintain (e, a)-synopses. However, in each instance care must be taken to ensure that the error guarantees specified in Section 1.2 hold for the modified algorithms.

We study the effect of adding time-sensitivity to two recent approximate frequency counting algorithms: *lossy*

counting [20] and the essentially identical algorithms of [8] and [17], which we refer to as *majority* counting.* The two algorithms (lossy counting and majority+ counting) use slightly different, although not unrelated, techniques to compute an (e, l)-synopsis over a single stream. In lossy counting, all frequency counts in the synopsis are periodically decremented by 1. The period of time between decrement operations is carefully chosen so that the resulting synopsis is guaranteed to be an (e, l)-synopsis. In contrast, in majority"1" counting all frequency counts in the synopsis are decremented by 1 whenever the synopsis size (measured in terms of the number of frequency counts) exceeds a predetermined threshold that depends on e. It has been shown analytically that majority"1" counting produces a (e, 1)-synopsis [8,17]. In Appendix A.I we prove that by adding exponentially decaying weighting to majority"1" counting we arrive at an algorithm that maintains an (e, a)-synopsis conforming to the error guarantees specified in Section 1.2.

Turning to lossy counting, it is relatively easy to show that by adding exponential weighting to lossy counting (and taking care to "catch up" by decrementing frequency estimates at epoch boundaries) we achieve a correct algorithm for maintaining an (e, a)-synopsis; we omit the simple proof (The modified algorithm is provided in Appendix A.2.). However, analysis of the space bound of the resulting synopsis is nontrivial. In Appendix A.2 we show that a time-independent space bound proportional to the logarithm of the maximum stream rate holds. Hence, the maximum size of an exponentially decayed lossy counting synopsis does not increase over time as long as the stream rate remains steady. In contrast, in the original lossy counting approach (i.e., using $a = 1$), the synopsis can grow logarithmically with time.

3 Finding Frequent Items in Distributed Streams

In this section we show how to maintain approximate time-sensitive frequency counts for frequent items in a distributed setting, and study how to set the precision gradient so as to minimize communication. Recall that in our scenario, m monitor nodes $M_1, M_2, \ldots, M_m$ relay data periodically, once every T time units, to a central root node R. Data may be relayed through a hierarchy of nodes interposed between the monitor nodes and the central root node, as illustrated in Figure 1. Let $I \geq 2$ denote the number of levels in the hierarchy. We number the levels from root to leaf, with the root node R of the communication hierarchy representing level 0, its children representing level 1, etc., and the monitor nodes $M_1, \ldots, M_m$ representing level $(I - 1)$. Let $d \geq 2$ denote the fanout of all non-leaf nodes in the hierarchy, i.e., the number of child nodes relaying data to each internal node.[2]

In this hierarchical communication structure, we associate with each non-root level $1 \leq i \leq (I - 1)$ of the communication hierarchy an error tolerance e_i. For correctness it must be ensured that $e_1 \geq e_2 \geq \ldots \geq e_{I-1} \geq 0$, which gives rise to a *precision gradient* along the communication hierarchy.[3] Any values of $e_1, \ldots, e_I$ satisfying the above constraints can be used, and the guarantees of Section 1.2 will hold. The manner in which the precision gradient (i.e., $e_1, \ldots, e_{I-1}$ values) is set determines the size of the synopsis that must be stored persistently at R, as well as the amount of communication that must be performed during frequency counting. For now, let us assume that some precision gradient has been decided upon. We return to the issue of how best to set the precision gradient in Section 3.1.

Given a precision gradient, our procedure for computing time-sensitive frequency counts for items occurring frequently in $S = S_1 \cup S_2 \cup \ldots \cup S_m$ is as follows. Recall that time is divided into equal epochs of length T. During each epoch, each monitor node M_i invokes a single-stream approximate frequency counting algorithm [8,17,20] using error parameter e_i to generate an $(e_i, 1)$-synopsis for the portion of stream S_i seen so far during the current epoch. Each monitor node then sends its $(e_I, 1)$-synopsis to its parent in the communication hierarchy, which combines the d $(e_{I-1}, 1)$-synopses it receives from its d children into a single $(e_{I-2}, 1)$-synopsis using Algorithm 1[4]. The same process is repeated until each of its children combines the d $(e_2, 1)$-synopses they receive into an $(e_1, 1)$-synopsis which is then sent to R.

The root node R maintains at all times a single (e, a)-synopsis S_A, from which the answer is derived. When, at

[2] For simplicity we assume all internal nodes of the communication hierarchy have the same fanout.

[3] For simplicity we assume that all nodes at the same level in the hierarchy use the same error tolerance.

[4] Algorithm 1 is based on lossy counting [20]. Alternatively, an algorithm based on majority"1" counting [8,17], in which e is substituted by $(e_i - e_{j+1})$, can be used for the same purpose. It can easily be shown that the corresponding algorithm based on majority"1" counting never results in smaller synopses than Algorithm 1.

the end of each epoch, *R* receives *d* (ei, l)-synopses from its children, *R* updates *SA* using Algorithm 2[5]. Then, *R* generates the new answer to be output for the current epoch by finding items in *SA* whose approximate count in *SA* exceeds *(s* — e) *-SA*n.*

Algorithm 1: Combine synopses from children (executed by nodes other than leaves and root)

Inputs: d (e*+i, *l)-synopses Si, S2,* - -, *Sd*
Output: single (e*, *1)-synopsis S*

1. Set <S:n:= $\sum_{j=1}^{d} \mathcal{S}_j{:}n$
2. For each *ue* $\bigcup_{j=1}^{d} S_{J9}$ set $S{:}\hat{c}(u) := \underset{j=1}{Yl}\ Sji\hat{c}(u)$
3. For each *u* G *S,* set $S{:}\hat{c}(u) := S{:}\hat{c}(u)$ — (e^ - e;+i)-5:n

---

Algorithm 2: Update the answer synopsis (executed at the root node *R)*

Input: d (ei, 1)*-synopses* S i, . . . , S^, S^
Output: new answer (e, *a)-synopsis SA*

1. For each n G S^, set $SA{:}\hat{C}(W) := a\text{-}SA'\hat{c}(u)$
2. SetS^:n := a-S^:n
3. For each uG $\bigcup_{j=i}^{<i}$ S7-, set $S^{\wedge}{:}\hat{c}(u) := «S^{\wedge}{:}\hat{c}(w) + E_{j=1}^{\wedge}Sj{:}\hat{c}(u)$
4. Set $\mathcal{S}_A{:}n := \mathcal{S}_A{:}n + \Sigma_{j=1}^{d}\mathcal{S}_j{:}n$
5. For each *u* G SA, set $SA'\text{-}\hat{C}(U) := SA\text{-}\hat{C}(U)$ — (e - ei)-Ef_1S,-:n

---

3.1 Setting the Precision Gradient

Our approach is to first set *e*\ based on space considerations at node *R* (using worst-case analysis), and then set the remaining error tolerance values e2, . . . , *t*\-\ so as to minimize communication.

The value of ei determines the maximum size of the synopsis *S^ that must be stored by node *R* at all times. Assuming a gradual precision gradient is used such that ei < e, analysis of the maximum size of *SA* is similar to the analysis of [20] and our analysis in [19] of time-sensitive lossy counting over a single-stream. If no time-sensitivity is employed (a = 1), the size of *SA* is at most $\frac{\ln((\frac{e}{\epsilon^{\wedge}})\frac{5\wedge\cdot n}{1})}{\cdot}$ counts (formula adapted from [20]); for $a < 1$, the size is at most $\frac{(i+g\ €i)\ (3+in(2\ /c\ /3+fc))}{\epsilon-\epsilon_1}$ counts? where /? = $[\log_{\alpha} A\ (1 + 7\wedge 7)1 + 1$ and /c denotes the maximum number of item occurrences on any input stream during any single epoch. Using *e*\ < e, the synopsis *SA* does not grow with

[5]Algorithm 2 is based on lossy counting [20]. Alternatively, an algorithm based on majority[4]" counting [8,17], in which e is substituted by (e — ei), can be used for the same purpose. The space used by such an algorithm is always l/(c — €1). However, as pointed out in [20], for common input streams (in particular, when input streams are Zipfian), the space requirement for an algorithm based on lossy-counting is significantly smaller than l/(e - ci).

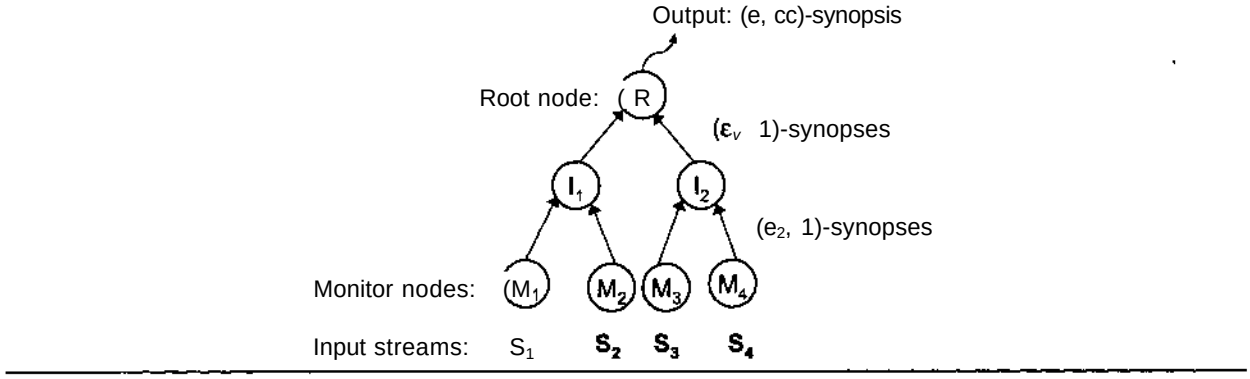


Figure 2: Example topology.

Table 2: Communication loads in example scenario.

ϵ_2	*Load on root node R*	*Maximum load on any link excluding links to R*	*Maximum load on any link*
0	*2*	27	27
0.03	*2*	14	14
0.05	54	14	27

time after reaching a steady-state size,as long as stream rates remain steady. In contrast, when $e\backslash = e$ (as in strategy SS2), the space requirement is much greater as we demonstrate empirically in Section 4.3. Our approach is to set ei such that the worst-case size of *SA* (under the maximum possible stream rate *k)* is below any space constraint at *R.*

Given a value for $e_{1?}$ the remaining error tolerance values €2,…, €/_i making up the precision gradient determine the communication load incurred. We illustrate the effect of the precision gradient on communication using the following rather contrived but simple example that highlights the effect clearly; our experimental results presented later in Section 4 are conducted over real-world data.

3.1.1 Motivating Example

Figure 2 shows the communication topology we use for our example. Suppose the overall user-specified error tolerance e = 0.05, and for simplicity assume $e\backslash$ w e = 0.05. Suppose that during one epoch 100 items occur on each of *Si*, £2, *S3* and 54, drawn from a universe of 27 distinct items. For ease of comprehension, we partition the 27 distinct items into three categories: A, B, and C. Category A contains one item and categories B and C each contain 13. The

Table 3: Link loads in example scenario.

	Mi ->/i *and* M3 —> *I2*		*M2-+I1 and M4 —> I2*		/1 -* *R and* I_2 -> *R*	
ϵ_2	*category*	*frequency estimate*	*category*	*frequency estimate*	*category*	*frequency estimate*
0	A	9	A	9	A	8
	B	6	B	1		
	C	1	C	6		
0.03	A	6	A	6	A	8
	B	3	C	3		
0.05	A	4	A	4	A	8
	B	1	C	1	B	1
					C	1

frequency of occurrence in each input stream of items in each category is given in the shaded region of Table 3. The single item in category A occurs nine times in each of Si, £2, S3 and S4. Each item in category B occurs six times each in Si and S3 but only once each in S_2 and S_4. The opposite is true for items in category C: each occurs once in each of Si and S3 but six times in each of S2 and S4.

Table 2 summarizes the effects of varying e_2, which determines the amount of error introduced at level 2 (nodes Mi — M_4), assuming lossy counting with per-epoch batch processing is used to produce the initial synopses at the leaf nodes. Three measures of communication load are reported: (1) load on the root node *R,* (2) maximum load on any link excluding links to *R,* and (3) maximum load on any link. In all cases, communication load is measured in terms of the number of frequency counts transmitted during the epoch. Setting $e_2 = 0.05$ corresponds to simple strategy SS2 outlined in Section 1.3. (We do not report measurements for SSI, in which ei = 0 and €2 = 0, since communication load is higher than under any of our three example strategies under all three metrics.)

To understand how these numbers come about, consider Table 3, which shows, for each setting of e_2, shows the frequency estimate for items of each category sent along each link. In the case in which €2 = 0, the estimated counts sent from leaf nodes Mi — M_4 to nodes *I* and I_2 (shown with shaded background) are exact. All other values in Table 3 are underestimates. We focus on the case in which *€2* = 0.03 to illustrate how these underestimates are computed. At each leaf node, when €2 = 0.03 application of the lossy counting algorithm leads to undercounting of each item's frequency by e2*100 = 0.03-100 = 3. Hence, estimated counts transmitted in synopses from the leaf nodes Mi — M4 to nodes 7i and *I2* are less than their actual counts by 3; some counts fall below zero and are eliminated. Once these synopses are received at nodes *I* and *I2,* Algorithm 1 is invoked, in which synopsis counts received from leaf nodes are first combined additively, and then decremented by (ei — 62) • 200 = 0.02-200 = 4. For the single item in Category A, leaf nodes Mi and M2 each supply a count of 6 to node /1, for a combined count of 12, which is then decremented by 4 for a final estimated count of 8 to be sent to node *R.* Items in Categories B and C each have combined counts of 3 at /1, which fall below zero when decremented by 4 and thus are not transmitted to *R.*

From Table 2 we observe a tradeoff between communication load on the root node *R* and load on links not connected to *R.* Furthermore, in this particular case (although not always true in general), of our three example strategies, the strategy of using a gradual precision gradient (62 = 0.03) is best with respect to all three metrics. To see why, consider that if error tolerances are made large for levels of the communication hierarchy close to the leaves (in the most extreme case, by setting ej_i = e, as in SS2), some locally-infrequent items are eliminated early, thereby reducing communication near the leaves. However, an undesirable side-effect arises in the presence of items just frequent enough at one or more leaf nodes to survive elimination locally, but not frequent enough overall to exceed the error threshold (as with items in categories B and C in our example). Counts for such items may avoid being eliminated until very late (or, worse, may never be eliminated), thus resulting in increased communication near the root. Hence, there is a tradeoff between high communication among non-root nodes and heavy load on the root node *R.*

The best way to set the precision gradient depends on the application scenario. For some applications the most important criterion may be to minimize load on the root node *R* where the answers are generated, which may need to devote the majority of its resources to other critical tasks for the application, even if that means increased load on the nodes responsible for monitoring streams and merging synopses. For other applications, it is most important to minimize the maximum load on any link to ensure that large volumes of input data can be handled without overloading network resources.

Next, we study the optimization problem of how best to set the precision gradient to achieve one of two objectives: (1) minimize communication load on the root node *R,* or (2) minimize worst-case communication load on the most heavily-loaded link in the hierarchy. Communication load is measured in terms of the number of frequency counts transmitted during one epoch. We study each optimization objective in turn in Sections 3.1.2 and 3.1.3, and provide optimal settings of the error tolerances $e_2, \ldots,$ e/_i making up the precision gradient. Then, since real-world data is unlikely to exhibit worst-case behavior, in Section 3.1.4 we propose a variant that seeks to achieve low load on the most heavily-loaded link, under non-worst-case inputs for which estimated data distributions are available.

3.1.2 Minimizing Total Load on the Root Node

Setting $€i = 0$ for all $2 \leq i \leq I - 1$, whereby all decrementing and elimination of synopsis counts is performed by children of node *R,* minimizes communication load on the root node *R* under any input streams. We term this

precision gradient setting MinRootLoad.

Lemma 1 *Given a value for ϵ_1, for any input streams no values of $\epsilon_2, \ldots, \epsilon_{l-1}$ satisfying $\epsilon_1 \geq \epsilon_2 \geq \ldots \geq \epsilon_{l-1}$ result in lower total communication load on node R than the values $\epsilon_2 = \epsilon_3 = \ldots = \epsilon_{l-1} = 0$.*

Proof: Consider node X, an arbitrary child of the root node R. Let S_X denote the union of all streams arriving at the monitor nodes belonging to the subtree rooted at X during one epoch. Algorithm 1 ensures that, for any setting of $\epsilon_2, \ldots, \epsilon_{l-1}$, counts for all items v with frequency $c(v) \geq \epsilon_1 \cdot |S_X|$ are sent over the link from X to R (here, $|S_X|$ denotes the number of item occurrences in S_X). Using $\epsilon_2 = \epsilon_3 = \ldots = \epsilon_{l-1} = 0$, it is easy to see that an item u will be sent over the link from X to R only if $c(u) \geq \epsilon_1 \cdot |S_X|$. Therefore, this setting of $\epsilon_2, \ldots, \epsilon_{l-1}$ results in the smallest possible number of counts sent over the link from X to R. Since this property holds for any child X of R, strategy MinRootLoad minimizes the total communication load on R, for any input streams. □

3.1.3 Minimizing Worst-Case Maximum Load on Any Link

In this section we show how to set $\epsilon_2, \ldots, \epsilon_{l-1}$ to minimize the maximum load on any communication link, in the worst case over all possible input streams. Our analysis assumes lossy counting is used to generate the local synopsis at each monitor node. We assume the buffer each monitor node uses for lossy counting is large enough to store frequency counts of all items arriving on the input stream during any one epoch. As we later confirm in Section 4, this assumption poses no problem in practice, particularly if the epoch duration is small.

For our worst-case analysis, we extend the set of possible inputs in two minor ways:

1. The occurrence frequency of an item arriving on an input stream can be a rational number.
2. Associated with each item u is a weight $w_u \in [0, 1]$. In an epoch, at most one item occurrence per input stream can be an occurrence of an item of weight less than 1. The cost of transmitting the count of item u with weight w_u is w_u. In a synopsis, $\mathcal{S}{:}n = \sum c(u) \cdot w(u)$.

As will become clear later, both of these enhancements allow load on a link to be expressed as a continuous function, which in turn simplifies our worst-case analysis. Neither enhancement alters the worst-case input significantly. First, during an epoch, at most one item occurrence per input stream can have non-integral weight. Second, any input with fractional item frequencies can be transformed into an input with integral frequencies that yields identical results by multiplying each frequency by a large number, and dividing all answers produced by the same number.

For notational ease, we transform the problem of setting $\epsilon_2, \ldots, \epsilon_{l-1}$ to that of setting $\Delta_2, \ldots, \Delta_{l-1}$, where for all $2 \leq i \leq l-2, \Delta_i = \epsilon_i - \epsilon_{i+1}$ and $\Delta_{l-1} = \epsilon_{l-1}$. It is required that $\Delta_i \geq 0$ for all $2 \leq i \leq l-1$, and that $\Sigma_{i=2}^{l-1} \Delta_i \leq \epsilon_1$. Δ_i denotes the *precision margin* at level i, i.e., the difference between the error tolerances at level i and level $i+1$.

Let the vector $\overline{\Delta} = (\Delta_2, \Delta_3, \ldots, \Delta_{l-1})$. Let I denote the contents of all input streams $S_1, \ldots, S_m$ during a single epoch. Let $\mathcal{I}$ denote the set of all possible instances of I.

Given an input I, a communication hierarchy $\mathcal{T}$ (defined by degree d and number of levels l), and a setting of the precision gradient $\overline{\Delta}$, let w represent the maximum load on any link in the communication hierarchy:

$$w(I, \mathcal{T}, \overline{\Delta}) = \max_{k \in links(\tau)} \{load(k)\}$$

Worst-case load W is defined as:

$$W(\mathcal{T}, \overline{\Delta}) = \max_{I \in \mathcal{I}} \{w(I, \mathcal{T}, \overline{\Delta})\}$$

Given a communication hierarchy $\mathcal{T}$, the objective is to set $\overline{\Delta}$ such that the worst-case load $W(\mathcal{T}, \overline{\Delta})$ is minimized. We denote such a value of $\overline{\Delta}$ by $\overline{\Delta}_{wc}$, defined as:

$$\overline{\Delta}_{wc} = \underset{\overline{\Delta}}{argmin} \{W(\mathcal{T}, \overline{\Delta})\}$$

We first show that it is sufficient to consider a specific subset of all instances of the general problem for worst-case analysis. Then we find precision gradient values $\overline{A}$ values that cause the worst-case load under any of these instances to be minimal.

There exists a subset *XA* of the set of all input instances *X* such that for all instances *I el — 1A*, there exists an instance *V e IA* such that for any T, $\overline{A}$, *w(I* T, $\overline{A}$) >_*w(I>* T, $\overline{A}$). Hence, *XA* denotes the set of *worst-case inputs.* Instance / is a member of *XA* if and only if it satisfies each of the following three properties:

- **PI:** For any two input streams *Si* and *Sj,* there is no item occurrence common to both *S{* and *Sj.*
- P2: For any input stream S*, all items occurring in *Si* occur with equal frequency.
- P3: For any two input streams S; and *Sj,* both the number of item occurrences, and the number of distinct items, in *Si* and *Sj* are equal.

Lemma 2 *Forfixed* T *and* $\overline{A}$, *given any input instance* 7, *it is find possible to find an input instance I' € XA such that w(V,*T', $\overline{A}$) ≥ w(J,T, $\overline{A}$).

Our proof of Lemma 2 is rather involved, and is provided in Appendix B.

From Lemma 2 we know it is sufficient to consider the set *XA* for worst-case communication load. Hence, we can rewrite our expression for *W(T,* $\overline{A}$) as:

$$W(T,\sim K) = \max_{I \wedge \cdot I \wedge A} \{w(/,T, \overline{A})\}$$

Property P3 of *XA* implies that the total number of item occurrences at any leaf node is the same. Let *n* denote this number *(\Si\ = n* for all $1 \leq i \leq m$*).* Let ***tc(j)*** denote the total number of item occurrences arriving on streams monitored by at the leaf nodes of a subtree rooted at a node at level *j*. It is easy to see that $tc(j) = d^{\wedge l \sim j \sim \wedge} \cdot n$, where *I* is the number of levels in the communication hierarchy and *d* is the fanout of all non-leaf nodes. The next lemma shows that worst-case inputs induce a high degree of symmetry on the resulting synopses.

Lemma 3 *For any input instance I* E XA> *the following two properties hold for the d^(e₀;, 1)-synopses relayed by the d^j level-j nodes to their parents:*

- *No item is present in more than one synopsis.*
- *The estimated frequency counts corresponding to any two items, even ifpresent in two different synopses, have the same value.*

Proof: We prove Lemma 3 by induction on *j*.

Base Case (*j* = *I* — 1): First, any input instance from the set *XA* satisfies properties PI and P3. Furthermore, recall that we assume each leaf node buffers stream contents for an entire epoch before reducing counts using the lossy counting algorithm. Hence, each leaf node reduces the frequency estimate corresponding to each item by the same amount: *tc(l* — 1) • Aj_i. Thus, it is easy to see that Lemma 3 holds for *j = (I* — 1).

Induction Step: Assume the lemma holds for level *j*. At level *j* — 1, the frequency estimate for each item is reduced by the same amount, *tc(j* — 1) • Aj_i, in Step (3) of Algorithm 1 (for convenience, we use Ai to denote €i - X^=$\bar{2}^1$ A^. Therefore, Lemma 3 holds for level *j* - 1. •

Due to the high degree of symmetry formalized in Lemma 3, the count for each item is eliminated (due to being decremented and falling below zero) at the same level of the communication hierarchy. Let us call this level *x.* If all counts are dropped at the leaf level, then *x = I* — 1. If all counts are retained through the entire process and are sent to the root node *R* (level 0), then *x* = 0. Otherwise, all counts are dropped at some intermediate level $1 \leq x \leq I - 2$.

The most heavily loaded link(s) are the ones leading to level *x.* To see why, consider that no data is transmitted on subsequent links and previous links have lower load since data is spread more thinly (in any communication hierarchy T, the number of links between levels decreases monotonically as data moves from leaves to the root).

When synopses are combined at nodes of level *i* using Algorithm 1, the frequency count estimate of each item is decremented by the quantity *tc(i)* • A* (let Ai = *c* — S^$\sim\bar{2}^1$ A*). Hence, the true frequency count of any item occurring

on some input stream must be $C = S\,j\bar{=}i+i\,(tc(j) \bullet A^{\wedge}) + 6$, where S is a small quantity[6]. The number of items present in each input stream is thus g?[7]. Since synopses for $d^{l\sim 1\sim x}$ input streams are transmitted through a node at level x, the load on the most heavily loaded link(s) is $L(x) = d^{l\sim l\sim x} \bullet g$. Clearly, the maximum value of $L(x)$ is achieved when $5 \longrightarrow 0$. The expression for L(x) can be simplified to:

$$L(x) = \frac{1}{\__7}$$

Now, our expression for the worst-case load on any link can be reduced to:

$$W\{T,\sim K\} = \max_{x=0,1,\ldots,l-2} \{L(x)\}$$

We desire to minimize $W(T, \bar{A})$ subject to the constraints $A_2, \ldots, A/\_i \geq 0$ and $E^{\wedge\wedge}Aj \leq ei$. It is easy to show that this minimum is achieved when $L(0) = L(1) = \bullet\bullet\bullet = L(l - 2)$.

Solving for $A_2, \ldots, Aj\_i$, we obtain: $A\text{»} = ci \bullet \frac{\sim^{\wedge}4^{*\wedge}4+2}{2 \leq i \leq / - 2}$ and $Aj\_i = e_x \bullet \frac{}{ld\_f[\_dJr2}$. Translating to error tolerances, we set $a = ei - \frac{d+}{}\wedge J17\text{-}\frac{(l-1}{\sim^{\wedge}}$ for all $2 \leq \dot{z} \leq / - 1$. This setting of $e_2, \ldots, e/\_i$ minimizes the worst-case communication load on any link. [8] We term this strategy MinMaxLoad.WC.

3.1.4 Good Precision Gradients for Non-Worst-Case Inputs

Real data is unlikely to exhibit worst-case characteristics. Consequently, strategies that are optimal in the worst case may not always perform well in practice. In terms of minimizing the maximum communication load on any link, the worst-case inputs are ones in which the set of items occurring on each input stream are disjoint. When this situation arises, a gradual precision gradient is best to use (as shown in Section 3.1.3). Using a gradual precision gradient, some of the pruning of frequency counts is delayed until a better estimate of the overall distribution is available closer to the root, thereby enabling more effective pruning. In the opposite extreme, when all input streams contain identical distributions of item occurrences, there is no benefit to delaying pruning, and performing maximal pruning at the leaf nodes (as in strategy SS2) is most effective at minimizing communication. In fact, it is easy to show that SS2 is the optimal strategy for minimizing the maximum load on any link when all input streams are comprised of identical distributions; we omit a formal proof. (Note, however, that SS2 still incurs a high space requirement on the root node R since it sets ei = e.)

We posit that most real-world data falls somewhere between these two extremes. To determine where exactly a data set lies with regard to the two extremes, we estimate the commonality between input streams $Si,\ldots, S_m$ by inspecting an epoch worth of data from each stream. We compute a *commonality parameter* 7 G [0,1] as $7 = \frac{1}{m}\text{—}Yl^{\wedge}i\ \hat{P}_i^*$ where Gi and Li are defined over stream Si as follows. The quantity Gi is defined as the number of distinct items occurring in Si that occur at least $e\text{-}\backslash Si\backslash$ times in Si and also at least e-15| times in $S = S\backslash \cup S_2 \cup \bullet\bullet\bullet \cup S_m$, where $\backslash S\backslash$ denotes the number of item occurrences in S during the epoch of measurement. The quantity Li is defined as the number of distinct items occurring in Si that occur at least $c\text{-}\backslash Si\backslash$ times in Si. Hence, commonality parameter 7 measures the fraction of items frequent enough in one input stream to be included in a leaf-level synopsis by strategy *SS2* that are also at least as frequent globally (in the union of all input streams).

A natural hybrid strategy is to use a linear combination of MinMaxLoad_WC and SS2, weighted by 7. The strategy is as follows: $sete^* = (1\text{-}7)\text{-}(ei \bullet \frac{\triangle^{\wedge}J1V^{\wedge}}{\text{"}^{\wedge}}) + r(c)$ for $2 \leq i \leq (/-2)$, and $e/\_i = \{l\text{-}^{\wedge})'Ui\text{-}\frac{}{Id\ f\ d+2}\backslash + 7^*(e)$. We term this hybrid strategy MinMaxLoad_NWC (for non-worst-case). Commonality parameter 7 = 1 implies that locally frequent items are also globally frequent, and SS2 (modified to use $e\backslash < e$) is a good choice. Conversely, 7 = 0 indicates that MinMaxLoadJWC is a good choice. For $0 < 7 < 1$, a weighted mixture of the two strategies is best.

[6] Recall that we allow the frequency of an item to be a real number.

[7] More precisely, each stream contains L^J items of weight 1 each and one item of weight = -^ — [§J- Note that each input stream contains at most one item with weight less than 1, as stipulated earlier.

[8] Lastly, we note that MinMaxLoadJWC remains the optimal precision gradient setting for minimizing the worst-case communication load on any link even if different nodes on the same level can have different e values. We omit a formal proof.

Table 4: Summary of precision gradient settings studied.

Strategy	*Description*	*Section Introduced*
Simple Strategy I (SS1)	Transmits raw data to root node R	1.3
Simple Strategy (SS2)	Reduces data maximally at leaf nodes	1.3
MinRootLoad	Minimizes total load on root in all cases	3.1.2
MinMaxLoad_WC	Minimizes worst-case maximum load on any link	3.1.3
MinMaxLoad_NWC	Variant for achieving low load on most heavily-loaded link, under non-worst-case inputs	3.1.4

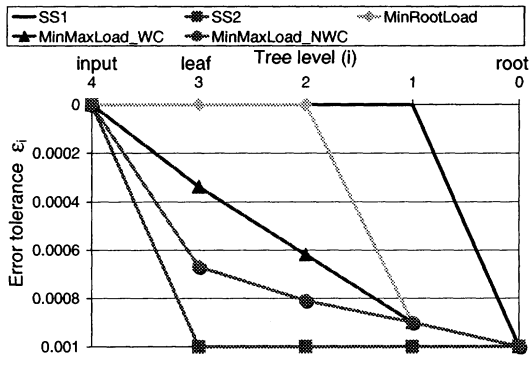


Figure 3: Precision gradients ($\epsilon = 0.001$; we assume $\gamma = 0.5$ for MinMaxLoad_NWC).

3.2 Summary

We now summarize the methods we introduced in Section 3.1 for setting the precision gradient, which consists of a set of error tolerance values $\epsilon \geq \epsilon_1 \geq \ldots \epsilon_{l-1} \geq 0$ associated with each level in the communication hierarchy. First, in all cases ϵ_1 should be set according to any space limitations at the root node R, which must persistently store a synopsis of recent item frequencies. The remaining error tolerances $\epsilon_2, \ldots, \epsilon_{l-1}$ may then be set so as to minimize the overall load on the root node or, alternatively, to minimize load on the most heavily-loaded communication link.

To minimize overall load on the root node, we introduced strategy MinRootLoad in Section 3.1.2, which delays all decrementing and elimination of frequency counts until immediately before data reaches the root node, in order to minimize the number of counts reaching the root. We also proposed two strategies aimed at minimizing communication load on the most heavily-loaded link: MinMaxLoad_WC (Section 3.1.3) and MinMaxLoad_NWC (Section 3.1.4). The former is guaranteed to minimize worst-case communication load on any link, whereas the latter aims to achieve low load in the presence of data that does not exhibit worst-case characteristics, and is therefore parameterized by certain high-level characteristics of the expected input data. Our strategies for setting the precision gradient are summarized in Table 4, and sample precision gradients are illustrated in Figure 3.

4 Experimental Evaluation

In this section we evaluate the performance of our newly-proposed strategies for setting the precision gradient, using the two naïve strategies suggested in Section 1 as baselines. We begin in Section 4.1 by describing the real-world data and simulated distributed monitoring environment we used. Then, in Section 4.2, we analyze the data using our model of Section 3.1.4 to derive appropriate parameters for our MinMaxLoad_NWC strategy that is geared toward performing in the presence of non-worst-case data. We report our measurements of space utilization on node R in

Table 5: Configurations of web application benchmarks.

Benchmark	*DBsize*	*Details*
AUCTION	489 MB	100,000 users 33,667 items
BBOARD	429 MB	500,000 users 213,292 comments

Section 4.3, and provide measurements of communication load in Section 4.4.

4.1 Data Sets

As described in Section 1, our motivating applications include detecting DDoS attacks and monitoring "hot spots" in large-scale distributed systems. For the first type of application, we used traffic logs from Internet2 [16], and sought to identify hosts receiving large numbers of packets recently. For the second type, we sought to identify frequently-issued SQL queries in two dynamic Web application benchmarks configured to execute in a distributed fashion.

The INTERNET2 [16] traffic traces were obtained by collecting anonymized netflow data from nine core routers of the Abilene network. Data were collected for one full day of router operation and were broken into 288 five-minute epochs. To simulate a larger number of nodes, we divided the data from each router in a random fashion. We simulated an environment with 216 network nodes, which also serve as monitor nodes.

For the web applications, we used Java Servlet versions of two publicly available dynamic Web application benchmarks: RUBiS [23] and RUBBoS [22]. RUBiS is modeled after eBay [9], an online auction site, and RUBBoS is modeled after slashdot [21], an online bulletin-board, so we refer them as AUCTION and BBOARD, respectively. We used the suggested configuration parameters for each application (given in Table 5), and ran each benchmark for 40 hours on a single node.We then partitioned the database requests into 216 groups in a round-robin fashion, honoring user session boundaries. We simulated a distributed execution of each benchmark with 216 nodes each executing one group of database requests and also serving as a monitor node.

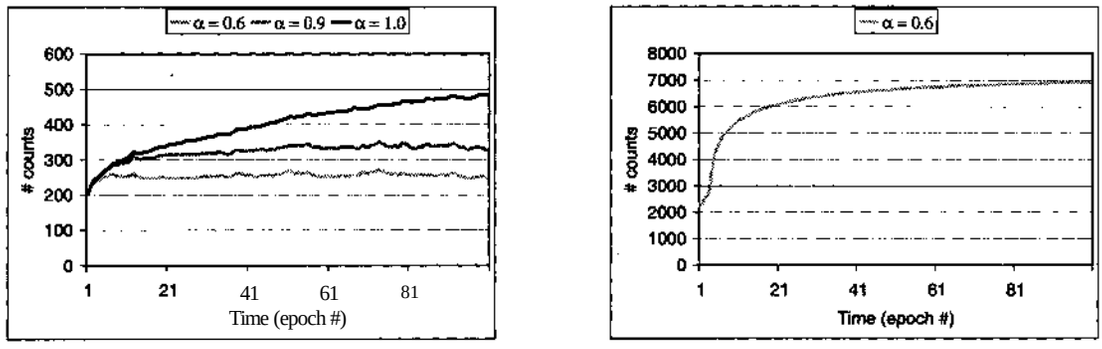
For all data sets, we simulated an environment with 216 monitoring nodes (m = 216) and a communication hierarchy of fanout six *(d* = 6). Consequently, our simulated communication hierarchy consisted of four levels including the root node (7 = 4). We set *s* = 0.01, e = 0.1 • s, and ei = 0.9 • e. Our simulated monitor nodes used lossy counting [20] in batch mode, whereby frequency estimates were reduced only at the end of each epoch (in all cases, less than 64KB of buffer space was used), to create synopses over local streams. The epoch duration T was set to 5 minutes for the INTERNET2 data set and 15 minutes for the other two data sets.

4.2 Data Characteristics

Using samples of each of our three data sets, we estimated the commonality parameter 7 for each data set. Recall that we use 7 to parameterize our strategy MinMaxLoad_NWC presented in Section 3.1.4. We obtained 7 values of 0.675 0.839 and 0.571 for the INTERNET2, AUCTION and BBOARD data sets respectively. Hence, the AUCTION data set exhibited the most commonality among all three data sets. Results presented in Section 4.4 show that AUCTION indeed has the most commonality.

4.3 Space Requirement on Root Node

Figure 4 plots space utilization at the root node *R* as a function of time (in units of epochs), for different values of the decay parameter a, using two different strategies for the precision gradient. The plots shown are for the INTERNET2 data set. The y-axis of each graph plots the current number of counts stored in the (e, a)-synopsis *SA* maintained by the root node *R.* Figure 4a plots synopsis size under our MinMaxLoad_WC strategy under three different values of *a:* 0.6, 0.9 and 1. As predicted by our analysis of Appendix A.I, when *a* < 1 the size of *SA* remains roughly constant after reaching steady-state, whereas when *a* = 1 synopsis size increases logarithmically with time (similar results were obtained for the non-distributed single-stream case). In contrast, when SS2 is used to set the precision



(a) MinMaxLoad-WC **(b) SS2**

Figure 4: Space needed at node *R* to store answer synopsis *SA

gradient (Figure 4b), the space requirement is almost an order of magnitude greater. This difference in synopsis size occurs because in SS2 frequency counts are only pruned from synopses at leaf nodes, so counts for all items that are locally frequent in one or more local streams reach the root node. No pruning power is reserved for the root node, and therefore no count in *SA* is ever discarded (irrespective of the *a* value). This result underscores the importance of setting 6i < e in order to limit the size of *SA*, as discussed in Section 3.1.

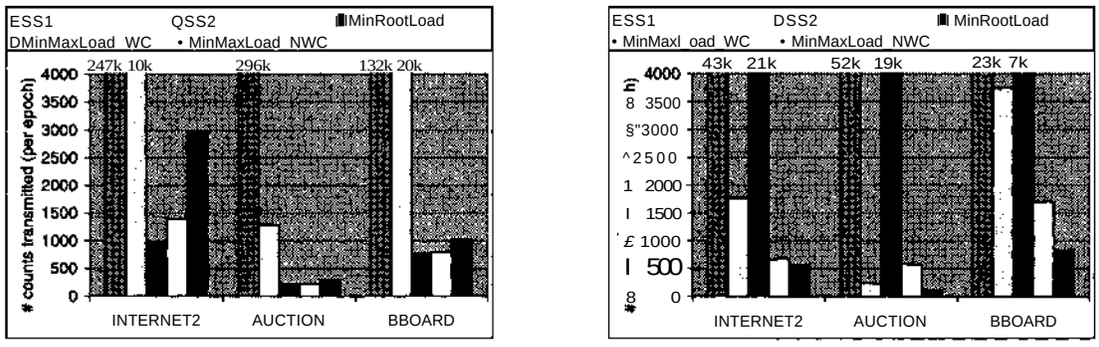
4.4 Communication Load

Figure 5 shows our communication measurements under each of our two metrics, for each of our three data sets, under each of the five strategies for setting the precision gradient listed in Table 4. First of all, as expected, the overhead of SSI is excessive under both metrics. Second, by inspecting Figure 5a we see that strategy MinRootLoad does indeed incur the least load on the root node *R* in all cases, as predicted by our analysis of Section 3.1.2. Under this metric, MinRootLoad outperforms both simple strategies SSI and SS2 by a factor of five or more in all cases measured. However, MinRootLoad performs poorly in terms of maximum load on any link, as shown in Figure 5b because no early elimination of counts for infrequent items is performed and, consequently, synopses exchanged near the leaf nodes tend to be quite large. As expected, MinMaxLoad_NWC performs best under that metric on all data sets. For the AUCTION data set, even though SS2 outperforms MinMaxLoad_WC (to be expected because of the high 7 value), our hybrid strategy MinMaxLoadJSfWC is superior to SS2 by over a factor of two. For the INTERNET2 and BBOARD data sets, the improvement over SS2 is more than a factor of three.

5 Related Work

Most prior work on identifying frequent items in streams, e.g., [8,17,20], only considers the single-stream case, and does not incorporate time-sensitivity. Recently, Arasu et al. [4] proposed a technique for finding frequent items in a sliding window over a single stream, which is a method of achieving time-sensitivity. Other recent work by Golab et al. [14] concentrates on specialized networking applications and proposes techniques for maintaining exact frequency counts over sliding windows, again over a single stream. In this paper we explore an alternative to sliding windows for achieving time-sensitivity, exponential weighting, and our primary focus is on distributed streams.

While we are not aware of any work on maintaining frequency counts for frequent items in a distributed stream setting, work by Babcock et al. [5] does address a related problem. In [5] the problem is to monitor continuously changing numerical values, which could represent frequency counts, in a distributed setting. The objective is to maintain a list of the top *k* aggregated values, where each aggregated value represents the sum of a set of individual



(a) Load on root node *R* **(b) Maximum load on any link**

Figure 5: Communication measurements ("k" denotes thousands).

values, each of which is stored on a different node. The work of [5] assumes a single-level communication topology and does not consider how to manage synopsis precision in hierarchical communication structures using in-network aggregation, which is the main focus of this paper.

6 Summary

In this paper we studied ways to extend algorithms for finding frequent items in a single data stream to incorporate time-sensitivity and work in a distributed setting. We began by analyzing the effect of applying exponentially decaying weighting to achieve time-sensitivity, and showed that the maximum space requirement becomes constant with respect to time.

We then turned to the problem of finding frequent items in the union of multiple distributed streams. The central issue is how best to manage the degree of approximation performed as partial synopses from multiple nodes are combined. We characterized this process for hierarchical communication topologies in terms of a precision gradient followed by synopses as they are passed from leaves to the root and combined incrementally. We studied the problem of finding the optimal precision gradient under two alternative and incompatible optimization objectives: (1) minimizing load on the central node to which answers are delivered, and (2) minimizing worst-case load on any communication link. We then introduced a heuristic designed to perform well for the second objective in practice, when data does not conform to worst-case input characteristics. Our experimental results on three real-world data sets showed that our methods of setting the precision gradient are greatly superior to naive strategies under both metrics, on all data sets studied.

References

[1] AGRAWAL, R., AND SRIKANT, R. Fast algorithms for mining association rules. In *Proceedings of the Twentieth International Conference on Very Large Data Bases* (1994).

[2] AKAMAI TECHNOLOGIES, I. Akamai, http://www.akamai.com/.

[3] AKELLA, A., BHARAMBE, A., REITER, M., AND SESHAN, S. Detecting DDoS attacks on ISP networks. In *Proceedings of the Twenty-Second ACM SIGMOD/PODS Workshop on Management and Processing of Data Streams* (2003).

[4] ARASU, A., AND MANKU, G. S. Approximate quantiles and frequency counts over sliding windows. In *Proceedings of the Twenty-Third ACM SIGACT-S1GMOD-SIGART Symposium on Principles of Database Systems* (2004).

[5] BABCOCK, B., AND OLSTON, C. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (2003).

[6] COHEN, E., AND STRAUSS, M. Maintaining time-decaying stream aggregates. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (2003).

[7] DATAR, M., GIONIS, A., INDYK, P., AND MOTWANI, R. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing* (2002).

[8] DEMAINE, E. D., LOPEZ-ORTIZ, A., AND MUNRO, J. I. Frequency estimation of internet packet streams with limited space. In *Proceedings of the Eleventh Annual European Symposium on Algorithms* (2003).

[9] EBAY INC. eBay, http://www.ebay.com.

[10] ESTAN, C, AND VARGHESE, G. New directions in traffic measurement and accounting. In *Proceedings of the ACM SIGCOMM 2002 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (2002).

[11] FANG, M., SHIVAKUMAR, N., GARCIA-MOLINA, H., MOTWANI, R., AND ULMANN, J. Computing iceberg queries efficiently. In *Proceedings of the Twenty-Fourth International Conference on Very Large Data Bases* (1998).

[12] GIBBONS, P., AND TIRTHAPURA, S. Estimating simple functions on the union of data streams. In *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures* (2001).

[13] GIBBONS, P. B., AND MATIAS, Y. New sampling-based summary statistics for improving approximate query answers. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data* (1998).

[14] GOLAB, L., DEHANN, D., DEMAINE, E., A LOPEZ-ORTIZ, AND MUNRO, J. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the Internet Measurement Conference* (2003).

[15] HAN, J., PEI, J., DONG, G., AND WANG, K. Efficient computation of iceberg queries with complex measures. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (2001).

[16] INTERNET2. Internet2 Abilene Network, http://abilene.internet2.edu.

[17] KARP, R. M., SHENKER, S., AND PAPADIMITRIOU, C. H. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst. 28,* 1 (2003), 51-55.

[18] KREYSZIG, E. *Advanced Engineering Mathematics,* Eighth ed. John Wiley and Sons, New York, 1999.

[19] MANJHI, A., SHKAPENYUK, V., DHAMDHERE, K., AND OLSTON, C. Finding (recently) frequent items in distributed data streams. Tech. Rep. CMU-CS-04-121, Carnegie Mellon University, 2004.

[20] MANKU, G., AND MOTWANI, R. Approximate frequency counts over data streams. In *Proceedings of the Twenty-Eighth International Conference on Very Large Data Bases* (2002).

[21] OPEN SOURCE DEVELOPMENT NETWORK INC. Slashdot. http://slashdot.org.

[22] RUBBoS. Rice University Bulletin Board System, http://rubbos.objectweb.org/.

[23] RUBlS. Rice University Bidding System, http://rubis.objectweb.org/.

A Analysis of Time-Sensitive Extensions

A.I Proof that Time-Sensitive Extension to Majority* Counting Ensures Error Guarantees

To make majority^{+} counting [8,17] time-sensitive, we extend it in a straight-forward fashion: each count in the synopsis is multiplied by a whenever a new epoch begins. For completeness, we provide the extended algorithm as Algorithm 3. We prove that this algorithm ensures the error guarantees specified in Section 1.2, i.e., for any item u_9 if, as before, $c(u)$ denotes the weighted frequency of occurrence of item u, and $S{:}\hat{c}(u)$ denotes its estimate produced by Algorithm 3, then $\{c(u) - e^{-\wedge 2_{weU}} c(w)) \leq S{:}\hat{c}(u)$. This inequality follows from the following lemma:

Lemma 4

$$Wu \pounds U, \quad (c(u) - S{:}\hat{c}(u)) \leq e^{-\underset{w \in U}{\wedge 2}} (^{c}(^{TM}) \sim S{:}\hat{c}(w))$$

Algorithm 3: Time-Sensitive Majority[4]" Counting —• maintaining (e, a)-synopsis *S*

Initially, *S:n* = 0

On arrival of a new item u:

1. If *S:c(u)* exists, set *S:c(u)* := *S:c(u)* + 1. Else, create *S:c(u);* set *S:c(u)* := 1
2. If \S\ ~> 1/e, for each *ueS,* set *S:c(u)* := <S:c(u) - 1; if 5:c(w) < 0, eliminate count *S:c(u)*
3. Set<S:n:=<S:n+l

On start of a new epoch:

1. For each *u* ***e*** *S,* set *S:c(u)* := *a-S:c(u)*
2. Set*S:n := a-*S:n

Proof: We prove this inequality by induction on *j,* the number of epochs completed (recall that an epoch consists of *T* time units). For each item, let *Cj(u)* and *S:ĈJ{U)* denote the actual and estimated weighted frequencies at the end of epoch *j*.

Base case 0 = 1)'. The left-hand side of the inequality represents the total number of times the count for item *u* was decremented by 1 during the first epoch. Each time item *u* was decremented (Step 3 of Algorithm 3), a total of |"^] counts (including w's count) were decremented by 1. Therefore, the total undercounting whenever *u* is decremented is at least (~~M–j~~M ?·. *j* \ ^ *s* quantity cannot exceed the total number of undercounting across all items $\sum_{weu}$ (ᶜiW ~ *Sicx̂iw))).* Hence, for epoch 1, *(d(u) - S:cî(u)) <_e>* £ *{ci{w) - S : c\(w)).*

Inductive step: Assume that Lemma 4 holds for epoch j , i.e., *(CJ(U) — S:CJ{U))* < *e-E*$_{weU}$*(cj(w) — S:CJ(W)).* By adapting the argument made in the base case *(j* = 1), we obtain:

$$\frac{1}{\epsilon}\left((c_{j+1}(u)-a\text{-}Cj(u))-(S:\hat{c}j+\hat{i}(u)-a\text{-}S:cj(u))\right) < a\left(\sum_{wet/} S:\overline{6j(w)} + \wedge \sum_{weu} (\overline{CJ+I(W)} - a>Cj\{w)) \; I_{/}\right) - \sum_{weu} 2\hat{\wedge}\text{«}S:\hat{C}j+i(u;) \quad (1)$$

The term *(CJ+I(U) — a-Cj(u))* on the left-hand side in the above inequality, is the number of times item *u* occurred in epoch *(j* + 1). Similarly, *(S:ĈJ+I(U) — a-S:Cj(u)))* represents the increase in item *u's* count during epoch *(j* + 1). Therefore, the difference of the two quantities, *((CJ+I(U) — a-Cj{u)) — (S:ĈJ+I(U) — a-S:ĈJ(U))),* represents the number of times the count for item *u* was decremented by 1 during epoch *j* + 1. Each time item *u* was decremented, a total of f^"| counts (including *u*\ count) were decremented. The left-hand side of Inequality (1) represents the minimum value of the total undercounting whenever *u* was decremented during epoch *(j -j-* 1). This quantity cannot exceed the total amount of undercounting across all items during epoch *(j* + *1),* the right-hand side of Inequality (1). The first term on the right-hand side is the initial sum of all counts at the end of epoch *j,* multiplied by a, plus the number of items that arrived during epoch (j +1). The second term is the sum of all counts at the end of epoch *(j* +1).

Combining Inequality (1) with the statement of the lemma for epoch *j,* we obtain:

$$\frac{1}{\epsilon}(c_{j+1}(u) - \mathcal{S}:\hat{c}_{j+1}(u)) \leq \sum_{weu}^{\wedge} (c_{j+1}(w) - \mathcal{S}:\hat{c}_{j+1}(w))$$

a

A.2 Derivation of Space Bound for Time-Sensitive Extension to Lossy Counting

In the original lossy counting algorithm [20], there is a single type of decrement operation: each count is decremented by 1 whenever \l/e] items arrive. In the time sensitive extension to lossy counting that we propose (for completeness,

Algorithm 4: Time-Sensitive Lossy Counting — maintaining (e, a)-synopsis *S*

Initially, *num* = 0, and *S:n* = 0
On arrival of a new item *u:*

1. If *S:ĉ(u)* exists, set *S:ĉ(u) := S:ĉ(u)* + 1 . Else, create *S:ĉ(u)*\ set *S:ĉ(u)* := 1

2. Set<S:n:=«S:n+l

3. Set *num := num* + **1**

4. If *num* = [i"|:

(a) Set num := 0

(b) For each *u* ***e*** 5, set «S:c(u) := 5:ĉ(u) - 1

(c) If *S:ĉ(u)* ≤ 0, eliminate count <S:ĉ(w)

On start of a new epoch:

1. For each *u* G *S*\

(a) Set <S:c(u) := *S:c(u)* - ^

(b) If *S:ĉ(u)* ≤ 0, eliminate count *S:ĉ(u)*

(c) Set5:ĉ(^) := *a-S:ĉ(u)*

2. Set *num* := 0

3. Set iS:n := a-«S:n

we provide the extended algorithm as Algorithm 4[9]), frequency estimates are reduced in value when either (a) a new epoch starts, or (b) [^1 items have arrived since the last count reduction operation. We provide detailed analysis of the worst-case space requirement for our time-sensitive extension to lossy counting. Before proceeding, we introduce some notation.

Let *Tj* = \{*(ui,ti) e S | j-T* ≤ *U* < *(j* + 1) *-T}*\ denote the number of items arriving on stream *S* in epoch *j*. Let $P = [\log_{ex}(1 + \frac{2}{\hat{}})1 + 1$ and let t_{now} denote the current time and let $p = [\hat{}\frac{t_{now}}{T}\hat{}J$ denote the number of epochs completed at time t_{now}. Let $k = \max_{j \le p}\{TJ\}$ denote the maximum number of items arriving on 5 in any epoch. We assume $a < 1$ throughout this section.

Lemma 5 *With k defined as above, the maximum weighted frequency count of any item is bounded from above as follows:* $\max_{u \in U}(c(w)) \le yz\hat{}\text{-}$

Proof: Recall from Section 1.2 that by definition, *c(u)* =]C<Mi,t<)€S,u<=u ^^ *. Using the definition of fe,

$$c(u) \le k' Y \hat{} a^{p-j} \le YZT\hat{}$$

As a result of the count reduction operations (mentioned at the start of this Section), any item occurrence within the current epoch results in a TTT reduction in the frequency count of each item. Since TTT ≥ y, any occurrence of

[9] Algorithm 4 can be modified in a straightforward way to achieve constant worst-case processing time.

an item within the current epoch results in the frequency count of each item being reduced by at least j ^ . Using this observation, we show in the next lemma that we can discount old occurrences of items from our calculations.

Lemma 6 *Assume a count reduction operation has just been carried out. If item u is not among the k-P most recent item occurrences in S, then S: c(u) = 0.*

Proof: Let /3± epoch boundaries occur during the last *k-/3* item occurrences. Since at most *k* items can appear on *S* in any epoch, $(5\backslash \geq \{(3 - 1)$.

Let $S{:}c^{f}(u)$ denote the value of item *us* count before the *k* •/? occurrences. There are $p\backslash$ epoch boundaries since item *u's* count was $S{:}\ c^{r}(u)\backslash$ and each epoch boundary scales down the count value by *a.* Moreover, arrival of any item *r* epochs before, results in a scaled-down decrement by a^{r}. Therefore, the minimum decrement due to *k-/3* item occurrences is ^ '2Zj=pl/?f~^{1}*(k '&^{p}~^{j})*- Therefore, $S{:}c(\hat{u})$ can be bounded from above as:

$$5 : \hat{C} (\leq) < / \wedge : \hat{C} ' (M) - \wedge . \overbrace{\quad}^{p-\beta_1+\beta-1} \} \wedge \quad k\text{-}a^{p\sim 3}$$
$$j=p-\beta_1$$

Using Lemma 5, the expression for $S{:}\hat{c}(u)$ becomes:

$$\begin{aligned} S{:}\hat{c}(u) &\leq a^{01}\frac{k}{1-a}\frac{k}{1+e}\frac{\epsilon}{;}\wedge \quad {}_{1-\beta+1}\cdot\frac{1-\alpha^{\beta}}{1-a} \\ &= (\alpha^{\beta_1-\beta+1})\cdot\frac{k}{1-\alpha}\cdot(\alpha^{\beta-1}-\frac{\epsilon}{1+\epsilon}\cdot(1-\alpha^{\beta})) \\ &\leq \frac{k}{1-\alpha}(\alpha^{\beta-1}\cdot\frac{2+\epsilon}{1+\epsilon}-\frac{\epsilon}{1+\epsilon}) \quad (\text{ since } a\text{^-^<}l \text{ and } a? \leq a?^{r1}) \\ &= 0 \quad (\text{substituting } a@\sim^{l} = \frac{\epsilon}{2+\epsilon}, \text{ follows from definition of } (3) \end{aligned}$$

Since $S{:}\hat{c}\{u) \leq 0$, $S{:}\hat{c}(u)$ must have been dropped from the synopsis. •

To help us analyze the maximum number of counts required, we associate a position with each item occurrence. We use this information to label the frequency counts present in the synopsis at time t_{now}. The label of a frequency count is *i* if it was created just after the i^{th} most recent item occurrence arrived. Let $d\{$ denote the number of counts in the synopsis with label *i.* Lemma 7 and Lemma 8 bound $d^{\wedge}$ which is eventually used to bound the maximum number of counts used by our time-sensitive extension to lossy counting.

Lemma 7 *Immediately after a count reduction operation is carried out,*

$$\frac{\epsilon}{1+\epsilon}\cdot\sum_{i=1}^{j}(i\cdot d_i) \leq j \quad \forall j \in \{1,2,\ldots\}$$

Proof: At t_{now}, Step (5) and 1(a) of Algorithm 4 ensure that each count with label *i* is decremented by at least $j^{\wedge}$ -*i.* Therefore, counts with labels up to *j* for any $j \in \{1, 2,\ldots\}$ are decremented by at least $^{\wedge} \cdot T, \sum_{i=1}^{J} i\text{-}di.$ If we consider the last *j* item occurrences, the maximum possible addition to all counts with labels at most *j* is *j.* Hence, $1fe\text{-}E?_{=}i(\dot{w}*i)\leq J^{\cdot} \quad Vje\{l,2,\ldots\}.$

•

Lemma 8 *Immediately after a count reduction operation is carried out,*

$$1$$
$$1 + {}^{6}{}_{i=l} \qquad i=l^{z}$$

Proof: This proof is similar to the proof in [20] for bounding the maximum number of counts that time-insensitive lossy counting maintains.

Base case ($j = 1$): Follows from Lemma 7.

Inductive step ($j = r$): Let Lemma 8 hold for $j = 1, 2, \ldots, r-1$. Adding Lemma 7 for $j = r$ to $(r-1)$ instances of Lemma 8 (one each for r varying from 1 to $r-1$) gives

$$\frac{\epsilon}{1+\epsilon} \cdot \Sigma_{i=1}^{r} i \cdot d_i + \frac{\epsilon}{1+\epsilon} \cdot \left(\Sigma_{i=1}^{1} d_i + \Sigma_{i=1}^{2} d_i + \cdots + \Sigma_{i=1}^{r-1} d_i\right) \leq r + \left(\Sigma_{i=1}^{1} \frac{1}{i} + \Sigma_{i=1}^{2} \frac{1}{i} + \cdots + \Sigma_{i=1}^{r-1} \frac{1}{i}\right)$$

which evaluates to $r \cdot \frac{\epsilon}{1+\epsilon} \cdot \Sigma_{i=1}^{r} d_i \leq r \cdot \Sigma_{i=1}^{r} \frac{1}{i}$. Thus, lemma 8 holds for $j = r$. □

Theorem 1 *The (ϵ, α)-synopsis maintained by our time-sensitive extension to the lossy counting algorithm contains at most $\frac{(1+\epsilon)\cdot(3+\ln(2\cdot k\cdot\beta+k))}{\epsilon}$ non-zero entries.*

Proof: We first analyze the maximum number of counts maintained by our time-sensitive extension to lossy counting at an instant just after a count reduction operation, as characterized at the beginning of this section.

Let x denote the number of item occurrences on S in the current epoch. Consider $2 \cdot k \cdot \beta + x$ most recent items. As before, let d_i denote the number of counts in the synopsis with label i. Let $D = \Sigma_{(j > 2\cdot k\cdot\beta + x)} d_j$.
Lemma 8 implies:

$$\sum_{i=1}^{j} d_i \leq \frac{1+\epsilon}{\epsilon} \cdot \sum_{i=1}^{j} \frac{1}{i} \; for \; j = 1, 2, \ldots, 2 \cdot k \cdot \beta + x - 1 \tag{2}$$

Any count in the synopsis with label greater than $2 \cdot k \cdot \beta + x$ must occur more than $(k \cdot \beta + x) \cdot \frac{\epsilon}{1+\epsilon}$ times in the last $2 \cdot k \cdot \beta + x$ items (This is because even if the value of each of the counts had the maximum possible value immediately after the $(2 \cdot k \cdot \beta + x)^{th}$ most recent element arrived, it would have decayed to zero by the time $(k \cdot \beta + x)^{th}$ most recent element arrived (Lemma 6), and after that the count value is decremented by $(k \cdot \beta + x) \cdot \frac{\epsilon}{1+\epsilon}$). Therefore, reasoning similar to Lemma 7, we get:

$$\frac{\epsilon}{1+\epsilon} \cdot \left(\sum_{i=1}^{2\cdot k\cdot\beta + x} i \cdot d_i + (k \cdot \beta + x) \cdot D \right) \leq 2 \cdot k \cdot \beta + x$$

By rearranging, we arrive at:

$$\sum_{i=1}^{2\cdot k\cdot\beta + x - 1} i \cdot d_i + (d_{2\cdot k\cdot\beta + x} + \frac{D}{2}) \cdot (2 \cdot k \cdot \beta + x) \leq \frac{1+\epsilon}{\epsilon} \cdot (2 \cdot k \cdot \beta + x) \tag{3}$$

By induction step similar to that employed in proof of Lemma 8, we obtain:

$$\sum_{i=1}^{2\cdot k\cdot\beta + x - 1} d_i + (d_{2\cdot k\cdot\beta + x} + D/2) \leq \frac{1+\epsilon}{\epsilon} \cdot \sum_{i=1}^{2\cdot k\cdot\beta + x} \frac{1}{i} \tag{4}$$

Adding $\frac{1}{2\cdot k\cdot\beta + x}$ times Inequality (3) to Inequality (4), we obtain:

$$\text{Number of counts in the synopsis} = \sum_{i=1}^{2\cdot k\cdot\beta + x + 1} d_i \leq \frac{1+\epsilon}{\epsilon} \cdot (1 + \sum_{i=1}^{2\cdot k\cdot\beta + x} 1/i) \leq \frac{1+\epsilon}{\epsilon} \cdot (2 + \ln(2 \cdot k \cdot \beta + k)) \tag{5}$$

The above proof holds for instants immediately after a count decrement operation is performed. However, between any two consecutive count decrement operations, at most $\lceil \frac{1}{\epsilon} \rceil$ items can arrive. Therefore, at any time the number of counts in the synopsis can be at most $\frac{(1+\epsilon)\cdot(3+\ln(2\cdot k\cdot\beta+k))}{\epsilon}$, where $\beta = \lceil \log_{\frac{1}{\alpha}}(1 + \frac{2}{\epsilon}) \rceil + 1$. □

Note that this space bound is independent of time. Hence, when $\alpha < 1$ the worst-case space requirement does not increase with time as long as stream rates remain steady.

B Proof of Lemma 2

We prove Lemma 2 in three steps. First, we show how to transform any input instance / into an instance *V* that satisfies Property PI and has $w(I^f,T,\overline{A}) \geq w(I,T,\overline{A})$. Then, we show how to transform an input instance *V* satisfying Property PI into an instance J" so that it satisfies both Properties PI and P2 and has $w(I'', T, \overline{A}) \geq w(I', T, \overline{A})$. Lastly, we show how to transform an input instance *I"* satisfying Properties PI and P2 into an instance *V"* that satisfies all three Properties PI, P2 and P3 (hence, /"' G *IA)* and has $w\{I''^f, T, \overline{A}) \geq w(I'', T, \overline{A})$.

Step 1: For any input instance /, let the node that transmits on the most heavily loaded link be node *Z,* whose level we denote by *z.* Let T_z be the subtree of which node *Z* is the root. Let us partition the input instance / into two parts: (a) I_z denoting the part of / arriving at leaf nodes in *Tz,* and (b) *I-^z* denoting the rest of / after excluding I_z *(I = Iz* U *I-^z)-* Since T is a hierarchical communication structure, the load that *Z* sends on its outgoing link (recall that load is measured in terms of the number of counts transmitted) depends only on I_z. This observation leads naturally to a two-part procedure for transforming any input / into *V* such that instance /' satisfies Property PI and has $w(I\backslash T, \overline{A}) \geq w(I, T, \overline{A})$: (a) transform *Iz* into I'_z so that V_z satisfies Property PI and has $w(I'_z \cup I\text{-}.z,T, \overline{A}) \geq w(iVT, \overline{A})$, and (b) transform *I-,z* into $I'^{\wedge}_z$ so that $V_z \cup I^{\wedge}_z = V$ satisfies Property PI and has $w(I', T, \overline{A}) \geq w(I, T, \overline{A})$.

Step l(a):To ensure that no item occurs at more than one leaf node in *Tz,* for each item *u* that occurs at *j* different leaf nodes Mi, M2,..., *Mj* with frequency counts ci, C2,..., *Cj* respectively, we create *j* new items *u\, U2,..., Uj.* Instead of item *u,* item *Uk* is included in the input stream *Sk* with weight $Ck/\wedge 2_i$ Q and frequency ^ *C{.*

While this operation might reduce the load on links inside Tz, it does not change the load on any link outside *Tz.* Hence, the load on the link from node *Z* to its parent remains unchanged. Thus, the resulting input instance I'_z satisfies Property PI and has $w(I'_z \cup I_{-,z}, T, \overline{A}) \geq w(I, T, \overline{A})$.

Step l(b): The only two reasons why $I'_z \cup I\text{-}_{,z}$ might not satisfy Property PI are (a) an item that occurs in I'_z also occurs in *I-,z,* or (b) an item occurs at more than one leaf in *I-,z>* To remedy either of (a) or (b), any conflicting item in /-,£ can be renamed to a new item. Thus, the resulting input instance *V (= Iz* U *I'-,z)* satisfies Property PI.

As with Step 1, we proceed in two parts in Steps 2 and 3.

Step 2(a): I'_z satisfies Property PI. Consider an item *u* that arrives at a leaf node *Y* of subtree T_z. For any level *j,* node *Y* has an unique level-j ancestor. For each such level-j ancestor node *X,* let *tc(j)* denote the total number of item occurrences in the the leaf nodes of the subtree rooted at node *X.* Since *u* does not occur at any other leaf node (Property PI), it is part of the synopsis node *Z* sends to its parent if and only if its occurrence frequency at *Y* is greater than $y = E\mathfrak{f}_{=l\_x}Aj \bullet tc(j)$. Furthermore, the total number of item occurrences at node *Y* is *tc(l — 1).* Therefore, no more than *tc(l — 1)/(y + S)* items that traverse the link from node *Z* to its parent could have arrived at node *Y,* for small $S > 0$. To transform $'I_z$ into I'_z, we replace all item occurrences at node *Y* with *tc(l - l)/(y + S)* new items, each of frequency *(y + 5).* We repeat this procedure for all other leaf nodes of the subtree T_z. Input instance I'_z thus obtained satisfies Properties PI and P2, and $w(I'_z \cup I'^{\wedge}_z, T, \overline{A}) \geq w(I', T, \overline{A})$.

Step 2(b): The input $I^{\wedge}_z$ can easily be transformed so that $I''_z \cup I''_{-z}$ satisfies Properties PI and P2 by replacing each item that occurs at any leaf node in $I'_{-,z}$ with a new item with occurrence frequency equal to 0, for any constant *(/>.* The resulting input instance /" (= $I''_z \cup I''_{-z}$) satisfies Properties PI and P2.

Step 3 (a): I''_z satisfies properties PI and P2. We first prove the following hypothesis: If the input I''_z satisfies Properties PI and P2, the load on the link from a node *X,* an ancestor of *Z,* to its parent due to I''_z is maximal when all leaf nodes in *Tz* have the same number of item occurrences and the same number of distinct items. We prove this by induction on the height of T_z.

Base Case *(Z* **is a leaf node):** Induction hypothesis is trivially true.

Induction Step: Let Y_u Y_2,..., Y_d denote the *d* children of *Z,* and let each *Yj* contribute *pj* to the total count at node *X.* Assume that the hypothesis holds for each of Yi, Y_2,..., Y_d. Let us term the subtrees of which some Y_f is a root *child-subtrees.* Each child-subtree satisfies all three properties PI, P2 and P3. The load on the link from node *X* to its parent due to $''I_z$ is $\wedge_{=\overline{1\,A\,/}\wedge\overline{+\,A\,/}}$ where $A_r = \ddot{E}^l_j l\backslash_{+1} d^{\wedge 1\wedge\sim j} Aj$ and A depends on precision gradient of nodes between node *Z* and node *X.* This expression, $E^{\wedge}_{=1\,A} \frac{1}{\cdot\,\cdot\,\cdot\,\cdot\,\cdot}$ subject to *pi* 4- *P2* + • • • + *Pd* = 1 attains its maximum (it can be seen for example, using the technique of Lagrange Multipliers [18]) when all *pj* are equal. Hence, the induction hypothesis holds for node *Z.*

I'''_Z can be obtained by multiplying the frequency counts of each item in the subtree rooted at Y_j by d/w_j. The resulting instance I'''_Z satisfies all three properties P1, P2 and P3 and has $w(I'''_Z \cup I''_{\neg Z}, \mathcal{T}, \overline{\Delta}) \geq w(I'', \mathcal{T}, \overline{\Delta})$.

Step 3(b): Since I'''_Z satisfies Properties P1, P2 and P3, let t items occur with frequency c each, at any leaf node in $\mathcal{T}_Z$. We transform $I''_{\neg Z}$ into $I'''_{\neg Z}$ as follows: at any leaf node not belonging to $\mathcal{T}_Z$, we create t distinct items each with occurrence frequency c. Thus, the resulting input instance I''' $(= I'''_Z \cup I'''_{\neg Z})$ satisfies Properties P1, P2 and P3.

Finally, we note that if two items u and v with weights w_u and w_v in some input stream have the same frequency, we can replace them by a single item y with weight $w_y = w_u + w_v$ and same frequency as that of u and v. This transformation does not change the behavior of our algorithm or the load on any link. By applying this procedure repeatedly to I''', we can ensure that there is at most one item with weight less than 1 on each input stream. □