Finding (Recently) Frequent Items in Distributed Data Streams

Amit Manjhi[*] Vladislav Shkapenyuk Kedar Dhamdhere[†] Christopher Olston
Carnegie Mellon University
{manjhi, vshkap, kedar, olston}@cs.cmu.edu

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.*

1. Introduction

The problem of identifying frequently occurring items in continuous data streams has attracted significant attention recently [4, 9, 12, 14, 19, 22]. Potential applications include identifying large network flows [12], answering iceberg queries [13], computing iceberg cubes [17] and finding frequent itemsets and association rules [1]. However, nearly all prior 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. Data from all sources needs to be aggregated to arrive at the final result, as in the distributed streams model of [15].

2. Time sensitivity. Recent data is more important than older data.

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

1. Monitoring usage 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.

2. Detecting malicious activities in networked systems:

(a) Detecting worms. Previously unknown Internet worms can be detected by discovering that a large number of recent traffic flows contain the same bit string [20]. Distributed monitoring can reduce detection time.

(b) 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.

1.1 Problem Variants

Both applications outlined above require algorithms for identifying recently frequent items in the union of many distributed streams, and estimating the corresponding occurrence frequencies. In general, we can

[*]Supported by an ITR grant from the NSF.
[†]Supported by NSF ITR grants CCR-0085982 and CCR-0122581.

classify applications of frequent item identification into four categories, in terms of whether they require (a) time-sensitivity and (b) distributed monitoring capability. 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* [7], 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 $S_1, S_2, \ldots, S_m$, 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 $S_1, S_2, \ldots, S_m$, instead of over a single stream.

(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. (Some work conducted concurrently with our own [4,16] also addresses problems quite similar to Variants (2) and (3), but there are significant differences with our work; see Section 4 for further discussion.) Algorithms for time-insensitive frequent item identification over a single stream include those presented in [9,19,22]. It is straightforward to extend these algorithms to handle Variant (2), although the effect on the space bounds and error guarantees of the resulting algorithms in some cases is nonobvious.

Variants (3) and (4) present a larger challenge. As we will show, simple adaptations of existing frequent item identification algorithms to work in a distributed setting incur excessive communication. In this paper we present a new framework for distributed frequent item identification 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 [22]. 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 universe 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: $\max\{0, c(u) - \epsilon \cdot N\} \leq \hat{c}(u) \leq c(u)$.

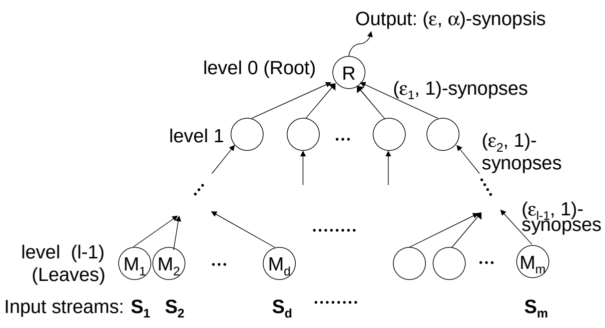


Figure 1: Hierarchical communication structure.

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

In the extended technical report version of this paper [21] we show how to extend two frequency counting algorithms that produce $(\epsilon, 1)$-synopses to produce (ϵ, α)-synopses, for any $\alpha \in (0, 1]$, to achieve Variant 2 of Section 1.1. In particular, we show how to do so for *lossy counting* [22] and the algorithm presented in both [9] and [19], which we refer to as *majority*$^+$ *counting*. We analyze the correctness and space requirements of the resulting algorithms. We show that the worst-case size of time-sensitive synopses is bounded by a time-independent constant.

1.3 Overview of Approach

There are two obvious, simple strategies for adapting single-stream frequency counting algorithms to a distributed setting to achieve Variants 3 and 4 of Section 1.1, and both have serious drawbacks:

SS1 (Simple Strategy 1): Periodically, at the end of every epoch, each monitor node M_i sends to the root node R the exact frequency counts of all items occurring in S_i 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 $\mathcal{T}$.

SS2: Each monitor node M_i maintains an $(\epsilon, 1)$-synopsis $\mathcal{S}_i$ over the recent portion of its local stream S_i. Intuitively, the $(\epsilon, 1)$-synopsis is a reduced summary of item frequencies that does not include items whose frequency in S_i is small. Periodically, at the end of every epoch, each M_i sends its local synopsis $\mathcal{S}_i$ to node R. Upon receiving all local synopses, node R combines them into a single unified $(\epsilon, 1)$-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 (ϵ, α)-synopsis containing estimated weighted counts from previous epochs, after multiplying those synopsis counts by α, to generate a new (ϵ, α)-synopsis valid for the current epoch. Lastly, items whose estimated time-decayed counts exceed the support threshold $\mathcal{T}$ (after taking into account the error tolerance) in this synopsis are output. [1]

Clearly, strategy SS1 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 ϵ 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 ϵ fraction of one or more individual streams but does not exceed ϵ 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 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

[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.

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 SS1), 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 Section 2 of 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.

In Section 3 we confirm our analytical findings of Section 2 through extensive experimental evaluation on three real-world data sets. Our experiments demonstrate that naïve methods of finding frequent items in distributed streams (SS1 and SS2) can incur high communication and storage costs compared with our methods. Related work is discussed in Section 4, and we summarize the paper in Section 5.

2. 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 $l \geq 2$ 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 $(l-1)$. 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 $1 \leq i \leq (l-1)$ of the communication hierarchy an error tolerance ϵ_i. For correctness it must be ensured that $\epsilon \geq \epsilon_1 \geq \ldots \geq \epsilon_{l-1} \geq 0$, which gives rise to a *precision gradient* along the communication hierarchy. (For now we assume that all nodes at the same level in the hierarchy use the same error tolerance.) Any values of $\epsilon_1, \ldots, \epsilon_{l-1}$ 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., $\epsilon_1, \ldots, \epsilon_{l-1}$ 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 2.1.

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, e.g., [9, 19, 22], using error parameter ϵ_{l-1} to generate an $(\epsilon_{l-1}, 1)$-synopsis for the portion of stream S_i seen so far during the current epoch. Each monitor node then sends its $(\epsilon_{l-1}, 1)$-synopsis to its parent in the communication hierarchy, which combines the d $(\epsilon_{l-1}, 1)$-synopses it receives from its d children into a single $(\epsilon_{l-2}, 1)$-synopsis using either Algorithm 1a (shown below; based on lossy counting [22]) or Algorithm 1b (shown below; based on majority$^+$ counting [9, 19]). The same process is repeated until each of R's children combines the d $(\epsilon_2, 1)$-synopses they receive into an $(\epsilon_1, 1)$-synopsis which is then sent to R.

The root node R maintains at all times a single (ϵ, α)-synopsis $\mathcal{S}_A$, from which the answer is derived. When, at the end of each epoch, R receives d $(\epsilon_1, 1)$-synopses from its children, R updates $\mathcal{S}_A$ using either Algorithm 2a (based on lossy counting) or Algorithm 2b (based on majority$^+$ counting). Then, R generates the new answer to be output for the current epoch by finding items in $\mathcal{S}_A$ whose approximate count in $\mathcal{S}_A$ exceeds $(s - \epsilon) \cdot \mathcal{S}_A\!:\!n$.

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

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

Inputs: d $(\epsilon_{i+1}, 1)$*-synopses* $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_d$
Output: single $(\epsilon_i, 1)$*-synopsis* $\mathcal{S}$

Algorithm 1a:

```
1. Set $\mathcal{S}{:}n := \sum_{j=1}^{d} \mathcal{S}_j{:}n$
2. For each $u \in \bigcup_{j=1}^{d} \mathcal{S}_j$, set $\mathcal{S}{:}\hat{c}(u) := \sum_{j=1}^{d} \mathcal{S}_j{:}\hat{c}(u)$
3. For each $u \in \mathcal{S}$, set $\mathcal{S}{:}\hat{c}(u) := \mathcal{S}{:}\hat{c}(u) - (\epsilon_i - \epsilon_{i+1})\cdot\mathcal{S}{:}n$
```

Algorithm 1b:

```
1. For each $\mathcal{S}_j \in \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d\}$ and for each $u \in \mathcal{S}_j$:
    (a) If $\mathcal{S}{:}\hat{c}(u)$ exists, set $\mathcal{S}{:}\hat{c}(u) := \mathcal{S}{:}\hat{c}(u) + \mathcal{S}_j{:}\hat{c}(u)$. Else, create $\mathcal{S}{:}\hat{c}(u)$; set $\mathcal{S}{:}\hat{c}(u) := \mathcal{S}_j{:}\hat{c}(u)$
    (b) If $|\mathcal{S}| \geq \frac{1}{\epsilon_i - \epsilon_{i+1}}$: let $u' := \underset{u \in \mathcal{S}}{argmin}\{\mathcal{S}{:}\hat{c}(u)\}$. For each $u \in \mathcal{S}$,
        set $\mathcal{S}{:}\hat{c}(u) := \mathcal{S}{:}\hat{c}(u) - \mathcal{S}{:}\hat{c}(u')$; if $\mathcal{S}{:}\hat{c}(u) \leq 0$, eliminate count $\mathcal{S}{:}\hat{c}(u)$
2. Set $\mathcal{S}{:}n := \sum_{j=1}^{d} \mathcal{S}_j{:}n$
```

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

Input: d $(\epsilon_1, 1)$*-synopses* $\mathcal{S}_1, \ldots, \mathcal{S}_d, \mathcal{S}_A$
Output: new answer (ϵ, α)*-synopsis* $\mathcal{S}_A$

Algorithm 2a:

```
1. Set $\mathcal{S}_A{:}n := \alpha\cdot\mathcal{S}_A{:}n + \Sigma_{j=1}^{d}\mathcal{S}_j{:}n$
2. For each $u \in \mathcal{S}_A$, set $\mathcal{S}_A{:}\hat{c}(u) := \alpha\cdot\mathcal{S}_A{:}\hat{c}(u)$
3. For each $u \in \bigcup_{j=1}^{d} \mathcal{S}_j$, set $\mathcal{S}_A{:}\hat{c}(u) := \mathcal{S}_A{:}\hat{c}(u) + \Sigma_{j=1}^{d}\mathcal{S}_j{:}\hat{c}(u)$
4. For each $u \in \mathcal{S}_A$, set $\mathcal{S}_A{:}\hat{c}(u) := \mathcal{S}_A{:}\hat{c}(u) - (\epsilon - \epsilon_1)\cdot\Sigma_{j=1}^{d}\mathcal{S}_j{:}n$
```

Algorithm 2b:

```
1. Set $\mathcal{S}_A{:}n := \alpha\cdot\mathcal{S}_A{:}n + \Sigma_{j=1}^{d}\mathcal{S}_j{:}n$
2. For each $u \in \mathcal{S}_A$, set $\mathcal{S}_A{:}\hat{c}(u) := \alpha\cdot\mathcal{S}_A{:}\hat{c}(u)$
3. For each $\mathcal{S}_j \in \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d\}$ and for each $u \in \mathcal{S}_j$:
    (a) If $\mathcal{S}_A{:}\hat{c}(u)$ exists, set $\mathcal{S}_A{:}\hat{c}(u) := \mathcal{S}_A{:}\hat{c}(u) + \mathcal{S}_j{:}\hat{c}(u)$. Else, create $\mathcal{S}_A{:}\hat{c}(u)$; set $\mathcal{S}_A{:}\hat{c}(u) := \mathcal{S}_j{:}\hat{c}(u)$
    (b) If $|\mathcal{S}_A| \geq \frac{1}{\epsilon - \epsilon_1}$, let $u' := \underset{u \in \mathcal{S}_A}{argmin}\{\mathcal{S}_A{:}\hat{c}(u)\}$. For each $u \in \mathcal{S}_A$, set $\mathcal{S}_A{:}\hat{c}(u) := \mathcal{S}_A{:}\hat{c}(u) - \mathcal{S}_A{:}\hat{c}(u')$; if $\mathcal{S}_A{:}\hat{c}(u) \leq 0$, eliminate count $\mathcal{S}_A{:}\hat{c}(u)$
```

2.1 Setting the Precision Gradient

Our approach is first to set ϵ_1 based on space considerations at node R (using worst-case analysis), and then set the remaining error tolerance values $\epsilon_2, \ldots, \epsilon_{l-1}$ so as to minimize communication.

The value of ϵ_1 determines the maximum size of the synopsis $\mathcal{S}_A$ that must be stored by node R at all times. If Algorithm 2b is used by the root node, the size of $\mathcal{S}_A$ is at most $\frac{1}{\epsilon - \epsilon_1}$ counts at all times. Otherwise, if Algorithm 2a is used, analysis of the maximum size of $\mathcal{S}_A$ is similar to the analysis of [22] and our own analysis in [21] of time-sensitive lossy counting over a single-stream, yielding the following results. If no time-sensitivity is employed ($\alpha = 1$), the size of $\mathcal{S}_A$ is at most $\frac{\ln((\epsilon-\epsilon_1)\cdot\mathcal{S}_A:n)}{\epsilon-\epsilon_1}$ counts (formula adapted from [22]); for $\alpha < 1$, the size is at most $\frac{(1+\epsilon-\epsilon_1)\cdot(3+\ln(2\cdot k\cdot\beta+k))}{\epsilon-\epsilon_1}$ counts, where $\beta = \lceil \log_{\frac{1}{\alpha}}(1 + \frac{2}{\epsilon-\epsilon_1}) \rceil + 1$ and k denotes the maximum number of item occurrences on any input stream during any single epoch. As long as stream rates remain steady, using $\epsilon_1 < \epsilon$, the synopsis $\mathcal{S}_A$ does not grow with time after reaching a steady-state size. In contrast, when $\epsilon_1 = \epsilon$ (as in strategy SS2), the space requirement increases with time as we demonstrate empirically in Section 3.3. Our approach is to set ϵ_1 such that the worst-case size of $\mathcal{S}_A$ (under the maximum possible stream rate k) is below any space constraint at R.

Given a value for ϵ_1 (such that $\epsilon_1 < \epsilon$), the remaining error tolerance values $\epsilon_2, \ldots, \epsilon_{l-1}$ 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 3 are conducted over real-world data.

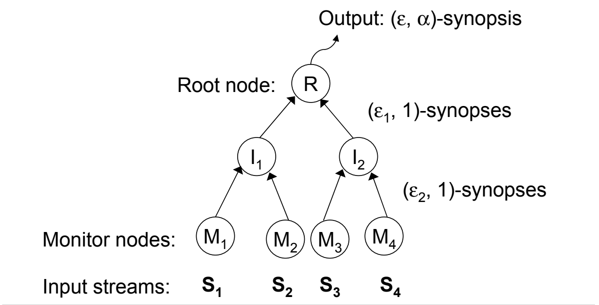


Figure 2: Example topology.

2.1.1 Motivating Example

Figure 2 shows the communication topology we use for our example. We assume Algorithm 1a is used at the intermediate nodes. Suppose the overall user-specified error tolerance $\epsilon = 0.05$, and for simplicity assume $\epsilon_1 \approx \epsilon = 0.05$. Suppose that during one epoch 100 items occur on each of S_1, S_2, S_3 and S_4, 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 frequency of occurrence in each input stream of items in each category is given in the shaded region of Table 2. The

Table 1: **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

Table 2: **Link loads in example scenario.**

	$M_1 \rightarrow I_1$ and $M_3 \rightarrow I_2$		*$M_2 \rightarrow I_1$ & $M_4 \rightarrow I_2$*		*$I_1 \rightarrow R$ & $I_2 \rightarrow R$*	
ϵ_2	*category*	*frequency estimate*	*cat.*	*freq. est.*	*cat.*	*freq. est.*
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

single item in category A occurs nine times in each of S_1, S_2, S_3 and S_4. Each item in category B occurs six times each in S_1 and S_3 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 S_1 and S_3 but six times in each of S_2 and S_4.

Table 1 summarizes the effects of varying ϵ_2, which determines the amount of error introduced at level 2 (nodes M_1 - - 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 $\epsilon_2 = 0.05$ corresponds to simple strategy SS2 outlined in Section 1.3. (We do not report measurements for SS1, in which $\epsilon_1 = 0$ and $\epsilon_2 = 0$, 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 2, which shows, for each setting of ϵ_2, the frequency estimate for items of each category sent along each link. In the case in which $\epsilon_2 = 0$, the estimated counts sent from leaf nodes M_1 - - M_4 to nodes I_1 and I_2 (shown with shaded background) are exact. All other values in Table 2 are underestimates. We focus on the case in which $\epsilon_2 = 0.03$ to illustrate how these underestimates are computed. At each leaf node, when $\epsilon_2 = 0.03$ application of the lossy counting algorithm leads to undercounting of each item's frequency by $\epsilon_2 \cdot 100 = 0.03 \cdot 100 = 3$. Hence, estimated counts transmitted in synopses from the leaf nodes M_1 - - M_4 to nodes I_1 and I_2 are less than their actual counts by 3; some counts fall below zero and are eliminated. Once these synopses are received at nodes I_1 and I_2, Algorithm 1a is invoked, in which synopsis counts received from leaf nodes are first combined additively, and then decremented by $(\epsilon_1 - \epsilon_2) \cdot 200 = 0.02 \cdot 200 = 4$. For the single item in Category A, leaf nodes M_1 and M_2 each supply a count of 6 to node I_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 I_1, which fall below zero when decremented by 4 and thus are not transmitted to R.

From Table 1 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 ($\epsilon_2 = 0.03$) 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 $\epsilon_{l-1} = \epsilon$, 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 select the precision gradient and synopsis-merging algorithm to use at each node, in order 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 2.1.2 and 2.1.3, and provide optimal algorithm choices and settings for the error tolerances $\epsilon_2, \ldots, \epsilon_{l-1}$ making up the precision gradient. Then, since real-world data is unlikely to exhibit worst-case behavior, in Section 2.1.4 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.

2.1.2 Minimizing Total Load on the Root Node

Using Algorithm 1a at all applicable nodes and setting $\epsilon_i = 0$ for all $2 \leq i \leq l-1$, whereby all decrementing and elimination of synopsis counts is performed by children of root node R, minimizes communication load on the root node R under any input streams. We term this strategy MinRootLoad.

Lemma 1 *Given a value for ϵ_1, for any input streams no values of $\epsilon_2, \ldots, \epsilon_{l-1}$ satisfying $\epsilon_1 \geq \epsilon_2 \geq \ldots \geq \epsilon_{l-1}$ and no choice of synopsis-merging algorithm results in lower total communication load on node R than the values $\epsilon_2 = \epsilon_3 = \ldots = \epsilon_{l-1} = 0$ and Algorithm 1a, assuming buffer space at each node is sufficient to store all inputs arriving during one epoch.*

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. Since an $(\epsilon_1, 1)$-synopsis is sent from X to R, 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 $\epsilon_2 = \epsilon_3 = \ldots = \epsilon_{l-1} = 0$ and Algorithm 1a at X, 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 $\epsilon_2, \ldots, \epsilon_{l-1}$ along with the use of Algorithm 1a 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. □

2.1.3 Minimizing Worst-Case Maximum Load on Any Link

In this section we show how to set $\epsilon_2, \ldots, \epsilon_{l-1}$ and how to select a synopsis-merging algorithm to use at each node so as to minimize the maximum load on any communication link, in the worst case over all possible input streams. We provide a two step solution. First, we show that for any precision gradient $\epsilon_2, \ldots, \epsilon_{l-1}$, use of Algorithm 1a at each node minimizes the load on every link, provided buffer space at each node is sufficient to store all inputs arriving during one epoch. Then, we derive the optimal precision gradient when Algorithm 1a is used at each node.

We begin with the issue of selecting a synopsis-merging algorithm.

Observation 1 *If, presented with identical inputs, Algorithm 1b produces output $\mathcal{S}$ and Algorithm 1a produces output $\mathcal{S}'$, then $\mathcal{S} : n = \mathcal{S}' : n$ and for all items $u \in \mathcal{S}$, $\mathcal{S} : \hat{c}(u) \geq \mathcal{S}' : \hat{c}(u)$.*

Observation 2 *Consider two sets of inputs to one of Algorithm 1a or Algorithm 1b. Let $input_1 = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d\}$, and $input_2 = \{\mathcal{S}'_1, \mathcal{S}'_2, \ldots, \mathcal{S}'_d\}$ where for all j ($1 \leq j \leq d$), $\mathcal{S}_j : n = \mathcal{S}'_j : n$ and for all items $u \in \mathcal{S}'_j$, $\mathcal{S}_j : \hat{c}(u) \geq \mathcal{S}'_j : \hat{c}(u)$. Let $input_1$ lead to output $\mathcal{S}$, whereas $input_2$ lead to output $\mathcal{S}'$. Then $\mathcal{S} : n = \mathcal{S}' : n$ and for all items $u \in \mathcal{S}$, $\mathcal{S} : \hat{c}(u) \geq \mathcal{S}' : \hat{c}(u)$.*

Lemma 2 *At any node X use of Algorithm 1a results in no higher communication on any link than use of Algorithm 1b.*

Proof: Follows from Observation 1 and multiple invocations of Observation 2. □

Lemma 3 *Given a choice between Algorithms 1a and 1b under any precision gradient, use of Algorithm 1a at each node minimizes the maximum load on any link.*

Proof: Follows from Lemma 2. □
It is trivial to extend this result to include leaf nodes, replacing Algorithm 1a with the original lossy counting algorithm.

Next, we show how to set $\epsilon_2, \ldots, \epsilon_{l-1}$ assuming Algorithm 1a is used at each node, and the lossy counting algorithm is used to generate the local synopsis at each monitor node. We also 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 3, 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 positive real 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, $\mathcal{S} : n = \sum w_u \cdot c(u)$.

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 real-valued item frequencies can be transformed into an input with nearly 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:

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

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 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 $\overline{\Delta}$ values that cause the worst-case load under any of these instances to be minimal.

There exists a subset $\mathcal{I}_{wc}$ of the set of all input instances $\mathcal{I}$ such that for all instances $I \in \mathcal{I} - \mathcal{I}_{wc}$, there exists an instance $I' \in \mathcal{I}_{wc}$ such that for any $\mathcal{T}$, $\overline{\Delta}$, $w(I', \mathcal{T}, \overline{\Delta}) \geq w(I, \mathcal{T}, \overline{\Delta})$. Hence, $\mathcal{I}_{wc}$ denotes the set of *worst-case inputs*. Instance I is a member of $\mathcal{I}_{wc}$ if and only if it satisfies each of the following three properties:

P1:For any two input streams S_i and S_j, there is no item occurrence common to both S_i and S_j.

P2: For any input stream S_i, all items occurring in S_i occur with equal frequency.

P3: For any two input streams S_i and S_j, both the number of item occurrences, and the number of distinct items, in S_i and S_j are equal.

Lemma 4 *For fixed $\mathcal{T}$ and $\overline{\Delta}$, given any input instance I, it is possible to find an input instance $I' \in \mathcal{I}_{wc}$ such that $w(I', \mathcal{T}, \overline{\Delta}) \geq w(I, \mathcal{T}, \overline{\Delta})$.*

Proof: Our proof of Lemma 4 is rather involved, and is provided in [21]. □

From Lemma 4 we know it is sufficient to consider the set $\mathcal{I}_{wc}$ for worst-case communication load. Hence, we can rewrite our expression for $W(\mathcal{T}, \overline{\Delta})$ as:

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

Property P3 of $\mathcal{I}_{wc}$ implies that the total number of item occurrences at any leaf node is the same. Let n denote this number ($|S_i| = n$ 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 $tc(j) = d^{(l-1-j)} \cdot n$, where l 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 5 *For any input instance $I \in \mathcal{I}_{wc}$, the following two properties hold for the d^j $(\epsilon_j, 1)$-synopses relayed by the d^j level-j nodes to their parents:*

1. *No item is present in more than one synopsis.*

2. *The estimated frequency counts corresponding to any two items, even if present in two different synopses, have the same value.*

Proof: See [21]. □

Due to the high degree of symmetry formalized in Lemma 5, 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 $x = l-1$. 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 $1 \leq x \leq l-2$.

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 $\mathcal{T}$, 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 $tc(i) \cdot \Delta_i$ (let

$\Delta_1 = \epsilon_1 - \Sigma_{i=2}^{l-1}\Delta_i$). Hence, the true frequency count of any item occurring on some input stream must be $C = \Sigma_{j=x+1}^{l-1}(tc(j) \cdot \Delta_j) + \delta$, where δ is a small quantity[3]. The number of items present in each input stream is thus $\frac{n}{C}$[4]. Since synopses for d^{l-1-x} input streams are transmitted through a node at level x, the load on the most heavily loaded link(s) is $L(x) = d^{l-2-x} \cdot \frac{n}{C}$. Clearly, the maximum value of $L(x)$ is achieved when $\delta \to 0$. The expression for $L(x)$ can be simplified to:

$$L(x) = \frac{1}{\Sigma_{j=x+1}^{l-1}(\Delta_j \cdot d^{x-j+1})}$$

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

$$W(\mathcal{T}, \overline{\Delta}) = \max_{x=0,1,\ldots,l-2}\{L(x)\}$$

We desire to minimize $W(\mathcal{T}, \overline{\Delta})$ subject to the constraints $\Delta_2, \ldots, \Delta_{l-1} \geq 0$ and $\Sigma_{j=2}^{l-1}\Delta_j \leq \epsilon_1$. It is easy to show that this minimum is achieved when $L(0) = L(1) = \cdots = L(l-2)$.

Solving for $\Delta_2, \ldots, \Delta_{l-1}$, we obtain: $\Delta_i = \epsilon_1 \cdot \frac{d-1}{(l-2)\cdot(d-1)+d}, 2 \leq i \leq l-2$ and $\Delta_{l-1} = \epsilon_1 \cdot \frac{d}{(l-2)\cdot(d-1)+d}$. Translating to error tolerances, we set $\epsilon_i = \epsilon_1 \cdot \frac{(l-1-i)\cdot(d-1)+d}{(l-2)\cdot(d-1)+d}$ for all $2 \leq i \leq l-1$. This setting of $\epsilon_2, \ldots, \epsilon_{l-1}$ minimizes worst-case communication load on any link. We term this strategy MinMaxLoad_WC. Under this strategy, the maximum possible load on any link is $L_{wc} = \frac{(l-2)\cdot(d-1)+d}{d \cdot \epsilon_1}$ counts per epoch. Lastly, we note that MinMaxLoad_WC remains the optimal precision gradient even if nodes of the same level can have different ϵ values. Informally, since with worst-case inputs all incoming streams have identical characteristics, maximum link load cannot be improved by using non-uniform ϵ values for nodes at a given level; we omit a formal proof for brevity.

2.1.4 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 2.1.3). 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 $\epsilon_1 = \epsilon$.)

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 $S_1, \ldots, S_m$ by inspecting an epoch worth of data from each stream. We compute a *commonality parameter* $\gamma \in [0,1]$ as $\gamma = \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{G_i}{L_i}$, where G_i and L_i are defined over stream S_i as follows. The quantity G_i is defined as the number of distinct items occurring in S_i that occur at least $\epsilon \cdot |S_i|$ times in S_i and also at least $\epsilon \cdot |S|$ times in $S = S_1 \cup S_2 \cup \cdots \cup S_m$, where $|S|$ denotes the number of item occurrences in S during the epoch of measurement. The quantity L_i is defined as the number of distinct items occurring in S_i that occur at least $\epsilon \cdot |S_i|$ times in S_i. Hence, commonality parameter γ 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 γ. The strategy is as follows: set $\epsilon_i = (1-\gamma) \cdot \left(\epsilon_1 \cdot \frac{(l-1-i)\cdot(d-1)+d}{(l-2)\cdot(d-1)+d}\right) + \gamma \cdot (\epsilon)$ for $2 \leq i \leq (l-2)$, and $\epsilon_{l-1} = (1-\gamma) \cdot \left(\epsilon_1 \cdot \frac{d}{(l-2)\cdot(d-1)+d}\right) + \gamma \cdot (\epsilon)$. We term this hybrid strategy MinMaxLoad_NWC (for non-worst-case). Commonality parameter $\gamma = 1$ implies that locally frequent items are also globally frequent, and SS2 (modified to use $\epsilon_1 < \epsilon$) is a good choice. Conversely, $\gamma = 0$ indicates that MinMaxLoad_WC is a good choice. For $0 < \gamma < 1$, a weighted mixture of the two strategies is best.

2.1.5 Summary

The precision gradient strategies we have introduced are summarized in Table 3. Sample precision gradients are illustrated in Figure 3.

3. Experimental Evaluation

In this section we evaluate the performance of our newly-proposed strategies for setting the precision gra-

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

[4]More precisely, each stream contains $\lfloor \frac{n}{C} \rfloor$ items of weight 1 each, and one item of weight $= \frac{n}{C} - \lfloor \frac{n}{C} \rfloor$. Note that each input stream contains at most one item with weight less than 1, as stipulated earlier.

Table 3: Summary of precision gradient settings studied.

Strategy	*Description (Section Introduced)*
Simple Strategy 1 (SS1)	Transmits raw data to root node R (1.3)
Simple Strategy 2 (SS2)	Reduces data maximally at leaves (1.3)
MinRootLoad	Minimizes total load on root in all cases (2.1.2)
MinMaxLoad_WC	Minimizes worst-case maximum load on any link (2.1.3)
MinMaxLoad_NWC	Achieves low load on heaviest-loaded link, under non-worst-case inputs (2.1.4)

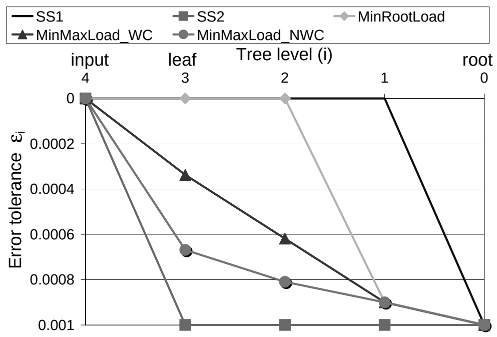


Figure 3: Precision gradients for $\epsilon = 0.001$, $\gamma = 0.5$.

dient, using the two naïve strategies suggested in Section 1 as baselines. We begin in Section 3.1 by describing the real-world data and simulated distributed monitoring environment we used. Then, in Section 3.2, we analyze the data using our model of Section 2.1.4 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 Section 3.3, and provide measurements of communication load in Section 3.4.

3.1 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 [18], 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 [18] 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 [10] and RUBBoS [10]. RUBiS is modeled after eBay [11], an online auction site, and RUBBoS is modeled after slashdot [23], an online bulletin-board, so we refer them as AUCTION and BBOARD, respectively. We used the suggested configuration parameters for each application, 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.

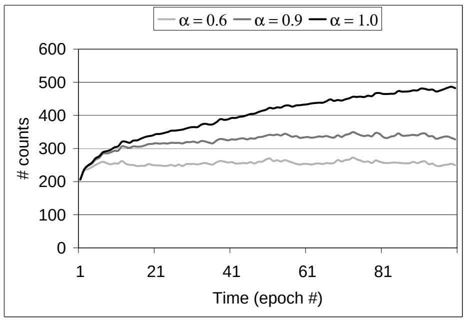
For all data sets, we simulated an environment with 216 monitoring nodes ($m = 216$) and a communication hierarchy of fanout six ($d = 6$). Consequently, our simulated communication hierarchy consisted of four levels including the root node ($l = 4$). We set $s = 0.01$, $\epsilon = 0.1{\cdot}s$, and $\epsilon_1 = 0.9{\cdot}\epsilon$. Our simulated monitor nodes used lossy counting [22] 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.

3.2 Data Characteristics

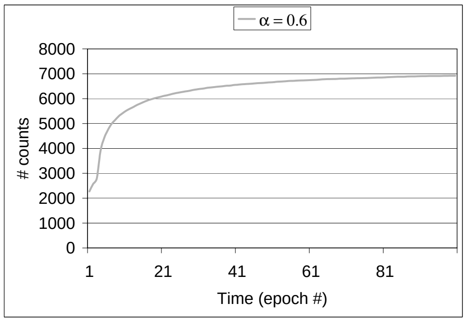
Using samples of each of our three data sets, we estimated the commonality parameter γ for each data set. Recall that we use γ to parameterize our strategy MinMaxLoad_NWC presented in Section 2.1.4. We obtained γ values of 0.675, 0.839 and 0.571 for the INTERNET2, AUCTION and BBOARD data sets respectively. Hence, the AUCTION data set exhibited the most commonality among all three data sets. Results presented in Section 3.4 show that AUCTION indeed has the most commonality.

3.3 Space Requirement on Root Node

Figure 4 plots space utilization at the root node R as a function of time (in units of epochs), using Algorithm 2a to generate the synopsis, for different values of the decay parameter α, 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 (ϵ, α)-synopsis $\mathcal{S}_A$ maintained by the root node R. Figure 4a plots synopsis size under our MinMaxLoad_WC strategy under three different values of α: 0.6, 0.9 and 1. As predicted by our analysis in [21], when $\alpha < 1$ the size of $\mathcal{S}_A$ remains roughly constant after reaching steady-state, whereas when $\alpha = 1$ synopsis size increases logarithmically with time (similar results were obtained for the

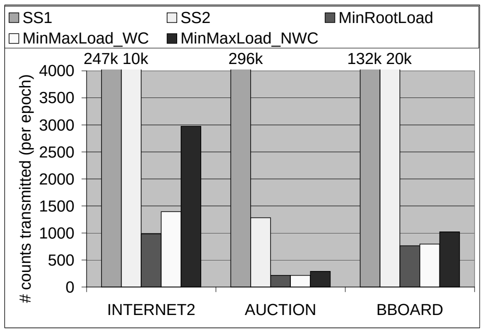


(a) MinMaxLoad_WC

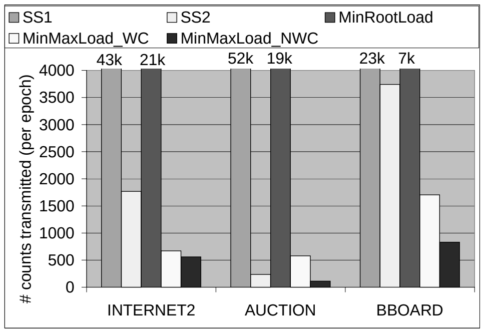


(b) SS2

Figure 4: **Space needed at node R to store answer synopsis $\mathcal{S}_A$.**



(a) Load on root node R



(b) Maximum load on any link

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

non-distributed single-stream case). In contrast, when SS2 is used to set the precision 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 $\mathcal{S}_A$ is ever discarded, irrespective of the α value. (The same situation occurs if Algorithm 2b is used instead of Algorithm 2a.) This result underscores the importance of setting $\epsilon_1 < \epsilon$ in order to limit the size of $\mathcal{S}_A$, as discussed in Section 2.1.

3.4 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 3. First of all, as expected, the overhead of SS1 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 2.1.2. Under this metric, MinRootLoad outperforms both simple strategies SS1 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 sent from the grand-children of the root node to the children of the root node 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 γ value), our hybrid strategy MinMaxLoad_NWC is superior to SS2 by a factor of over two. For the INTERNET2 and BBOARD data sets, the improvement over SS2 is more than a factor of three. On the negative side, total communication (not shown in graphs) is somewhat higher under MinMaxLoad_WC than under SS2 (increase of between 7.5% and 49.5%, depending on the data set).

4. Related Work

Most prior work on identifying frequent items in data streams [6, 8, 9, 19, 22] only considers the single-stream case. While we are not aware of any work on maintaining frequency counts for frequent items in a distributed stream setting, work by Babcock and Olston [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 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.

The work most closely related to ours is the recent work of Greenwald and Khanna [16], which addresses the problem of computing approximate quantiles in a general communication topology. Their technique can be used to find frequencies of frequent items to within a configurable error tolerance. The work in [16] focuses on providing an asymptotic bound on the maximum load on any link (our result adheres to the same asymptotic bound). It does not, however, address how best to configure a precision gradient in order to minimize load, which is the particular focus of our work.

5. Summary

In this paper we studied 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 naïve strategies under both metrics, on all data sets studied.

Acknowledgments

We thank Arvind Arasu and Dawn Song for their valuable input and assistance.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[2] I. Akamai Technologies. Akamai. `http://www.akamai.com/`.

[3] A. Akella, A. Bharambe, M. Reiter, and S. Seshan. Detecting DDoS attacks on ISP networks. In *PODS Workshop on Management and Processing of Data Streams*, 2003.

[4] A. Arasu and G. S. Manku. Approximate quantiles and frequency counts over sliding windows. In *PODS*, 2004.

[5] B. Babcock and C. Olston. Distributed top-k monitoring. In *SIGMOD*, 2003.

[6] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages and Programming*, 2002.

[7] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, 2003.

[8] G. Cormode and S. Muthukrishnan. What's hot and what's not: Tracking frequent items dynamically. In *PODS*, 2003.

[9] E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*, 2003.

[10] DynaServer. RUBis and RUBBos. `http://www.cs.rice.edu/CS/Systems/DynaServer/`.

[11] eBay Inc. eBay. `http://www.ebay.com`.

[12] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.

[13] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ulmann. Computing iceberg queries efficiently. In *VLDB*, 1998.

[14] P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD*, 1998.

[15] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *Symposium on Parallel Algorithms and Architectures*, 2001.

[16] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS*, 2004.

[17] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of iceberg queries with complex measures. In *SIGMOD*, 2001.

[18] Internet2. Internet2 Abilene Network. `http://abilene.internet2.edu`.

[19] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 2003.

[20] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th Usenix Security Symposium*, 2004.

[21] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. Technical report, 2004. `http://www.cs.cmu.edu/~manjhi/freqItems.pdf`.

[22] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, 2002.

[23] Open Source Development Network Inc. Slashdot. `http://slashdot.org`.