5-2007

Monitoring frequent items over distributed data streams.

Robert Harrison Fuller 1984-
University of Louisville

MONITORING FREQUENT ITEMS OVER DISTRIBUTED DATA STREAMS

By

Robert Harrison Fuller
B.S., Indiana University Southeast, 2005

A Thesis
Submitted to the Faculty of the
Graduate School of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Masters of Science

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

May 2007

MONITORING FREQUENT ITEMS OVER DISTRIBUTED DATA STREAMS

By

Robert Harrison Fuller
B.S., Indiana University Southeast, 2005

A Thesis Approved on

April 3, 2007

By the following Thesis Committee:

____
Dr. Mehmed Kantardzic, CECS (Thesis Director)

____
Dr. Anup Kumar, CECS

____
Dr. Julius Wong, ME

ACKNOWLEDGEMENTS

The author would like to acknowledge his thesis director Dr. Mehmed M. Kantardzic. This thesis would not be possible without his continued guidance, encouragement, and insight. The author would is also like to convey his sincere appreciation to the reading and examination committee members. The author would like to recognize USA Funds for providing financial support throughout his graduate studies. Finally, appreciation is given to the family members of the author for provided love and support throughout the author's life.

ABSTRACT

MONITORING FREQUENT ITEMS OVER DISTRIBUTED DATA STREAMS

Robert H. Fuller

April 3, 2007

Many important applications require the discovery of items which have occurred frequently. Knowledge of these items is commonly used in anomaly detection and network monitoring tasks. Effective solutions for this problem focus mainly on reducing memory requirements in a centralized environment. These solutions, however, ignore the inherently distributed nature of many systems. Naively forwarding data to a centralized location is not practical when dealing with high speed data streams and will result in significant communication overhead.

This thesis proposes a new approach designed for continuously tracking frequent items over distributed data streams, providing either exact or approximate answers. The method introduced is a direct modification to an existing communication efficient algorithm called Top-K Monitoring. Experimental results demonstrated that the proposed modifications significantly reduced communication cost and improved scalability.

Also examined in this thesis is the applicability of frequent item monitoring at detecting distributed denial of service attacks. Simulation of the proposed tracking method against four different attack patterns was conducted. The outcome of these experiments showed promising results when compared to previous detection methods.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO DATA STREAMS

1.1 Background

Recent years has shown a rapid increase in applications requiring the processing of data streams. Data streams are sequences of data that arrive continuously over time (Cohen and Strauss, 2004). Examples include network traffic data, stock tickers, click streams, and data generated by wireless sensors. In each of these examples the data stream may be represented as a sequence of tuples containing destination addresses, stock quotes, environmental readings, etc.

Data streams from a variety of application domains generally exhibit very similar properties which yield interesting challenges. First the data generally arrives at a very fast pace, sometimes as fast as several gigabytes a second (Lee and Ting, 2006). This requires real-time processing of each update tuple to keep pace with the rate of the stream. Second the final length of the stream is often times not known in advance. As a result, it is modeled as a never-ending or unbounded stream (Zhu and Shasha, 2002). Since processing must be done in real-time, storing to a secondary storage disk is not feasible. Additionally, only a limited amount of a main memory is available. Third the data is usually distributed in large networks (Sun, Papadimitriou, and Faloutsos, 2006). Thus communication must be limited to observe any imposed network constraints. Finally, processing results are generally given to the user continuously, always reflecting the current state of the data stream (Babcock and others, 2002).

A common data stream processing task is to find items in the data which have occurred frequently. An item is defined to be frequent if it accounts for a high percentage of the total number of occurrences seen so far. Important data stream applications of frequent item analysis include:

1. *Web Advertising:* Revenue may be increased by recognizing users who frequently click advertisements and displaying Pay-Per-Click advertisements when they visit your site (Metwally, Agrawal, and Abbadi, 2005).
2. *Network Flow Management*: Generally only a few flows will account for a large portion of bandwidth in a network. Knowing these flows can be used to allocate bandwidth more fairly (Stanojevic).
3. *Detecting Network Anomalies*: Some network attacks exhibit frequent characteristics. For example, worms can be detected by determining frequently occurring substring patterns in traffic flows (Kim and Karp, 2004). Another example includes the detection of distributed denial of service (DDoS) attacks. Recently, methods have shown that by identifying destination addresses which have received a large number of packets over a given time can be used to detect DDoS attacks (Akella and others, 2003; Manjhi and others, 2005; Sekar and others, 2006).

In this thesis we consider the problem of monitoring frequent items over distributed data streams. The term monitoring, means that the up-to-date list of frequent items are displayed to the user continuously in real-time. This problem inherits many of the data stream processing challenges described previously. Due to the unbounded nature and high data rates of streams, we must propose a method that is both space and time efficient. Additionally in this scenario, frequently occurring items must be determined from multiple data streams originating from dispersed sources. As a result, communication costs must be considered.

1.2 Formal Problem Statement

In the following subsections we describe the distributed architecture used in our monitoring approach. This architecture is the most commonly observed in prior work and represents a large number of real world networks (Babcock and Olston, 2003; Cormode and Garofalakis, 2005a; Garofalakis, 2005b; Cormode and others, 2005; Keralapura, Cormode, and Ramamirtham, 2006). Additionally, we will formalize our representation of the data streams observed at each distributed site. Finally, we will provide a formal definition to the frequent items problem and it variations. The notation introduced in these sections will be used throughout the remainder of this thesis.

1.2.1 System Model

The distributed environment used in our method has been defined as a single-level hierarchical architecture (Babcock and Olston, 2003). It consists of $m + 1$ nodes and m distributed data streams. Of the nodes, $N_1, N_2, \ldots, N_m$ are used for summarizing the m data streams and are called monitoring nodes. Node N_0 is a specialized coordinator node. The coordinator node is responsible for displaying the set of frequent items over the union of the m distributed data streams. As in previous work, communication is conducted amongst the monitoring nodes and the coordinator. There is no direct communication between any two monitoring nodes (Babcock and Olston, 2003; Cormode and Garofalakis, 2005a; Cormode and others, 2005). A schematic of this architecture can be seen in Figure 1.1.

Each of the distributed data streams S_1, S_2, ..., S_m, is used as input to corresponding monitoring nodes N_1, N_2, ..., N_m. The data streams consist of a sequence of tuples ordered by time of occurrence. Each tuple is of the form $\langle o_j, t_j \rangle$, where o_j is the unique identifier of a specific item of interest pulled from a finite (but possibly large) set of allowable identifiers U, and t_j is the timestamp of the tuple. Identifiers may be repeated any number of times in a data stream. An example of an input stream, corresponding to monitoring node N_1, may be $S_1 = \{\langle 2, 0.024 \rangle, \langle 2, 0.029 \rangle, \langle 1, 0.050 \rangle, \langle 0, 0.056, \rangle\}$ where U = {0, 1, 2, 3}.

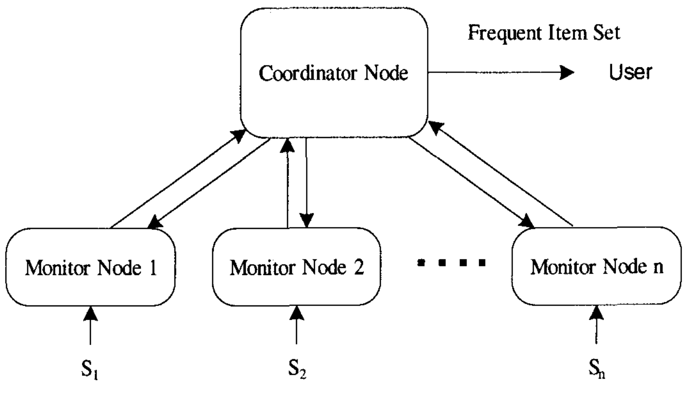


FIGURE 1.1 – Single-level hierarchical architecture.

As stated, each monitoring node maintains a summary of its corresponding data stream. This summary is made by managing a set of frequency counts $C_i = \{c_{1,i}, c_{2,i},\ldots, c_{n,i}\}$, where each $c_{j,i} \in C_i$ corresponds to an item identifier from the set U. Initially each frequency count is equal to zero, and for each input tuple $\langle o_j, t_j\rangle$ to N_i, $c_{j,i}$ is incremented by one. Therefore, each frequency count in the set C_i maintains the number of occurrences of an item in the data stream S_i on monitoring node N_i. To extend the previous example, $C_1 = \{1, 1, 2, 0\}$.

1.2.2 Frequent Items Problem

The purpose of the monitoring structure discussed above is to monitor frequent items over the union of the distributed data streams. Given an item o_j and corresponding counters $\{c_{j,1}, c_{j,2},\ldots, c_{j,m}\}$, we call o_j frequent if $\sum_{1\leq i\leq m} c_{j,i} \geq s\cdot N$, where $s \in (0, 1)$ is a user defined support parameter and N is the accumulative frequency of all observed items. The set of all frequent items F, therefore, contains all items which have occurred across the union of the m data streams more than $s\%$ of the total number of item occurrences.

1.2.3 Approximate Frequent Items Problem

Due to the unbounded nature of data streams it is impossible to store them in main memory or even secondary storage. This has motivated the creation of a variety of summary data structures which sacrifice correctness and provide approximate solutions (Li, Lee, and Shan, 2005). These summary data structures require only a limited amount of memory, but provide only approximate frequency counts. To solve the frequent items problem with approximate frequencies, an extension has been proposed called the ε-deficient frequent items problem. The definition for this problem we adopted comes from the work of Manku and Motwani (2002).

To extend the previous definition given in Section 1.2.2, the ε-deficient frequent items problem allows a degree of error on the frequency counts which is bounded by a user defined error tolerance parameter $\varepsilon \ll s$. The membership of an item in the set F is modified with the following requirements:

1. Those whose true frequency exceeds $s \cdot N$ are in the frequent item set.

2. No item whose true frequency is less than $(s - \varepsilon) \cdot N$ is in the frequent item set

3. Frequency counts are under counted or over counted by at most $\varepsilon \cdot N$.

The resulting membership test derived from these three points is determined by whether the frequency counts are over estimated or under estimated. If the items are under estimated, an item is called frequent if $\sum_{1 \le i \le m} c_{j,i} \ge (s - \varepsilon) \cdot N$, where each $c_{j,i}$ is an approximate frequency count underestimating the true frequency of o_j by at most $\varepsilon \cdot N$.

1.2.4 Top-K Elements Problem

The top-k elements problem is very similar to that of finding frequent items. Given a set of items $o_1, \ldots, o_n$ and corresponding frequency counts, we create a ranked list sorted by non-increasing frequency. We return the set which contains the k most occurring items, where k is a user defined parameter bounding the size of the list.

The top-k elements problem can additionally be used to determine frequent items. For a given support parameter s, there can be at most $\frac{1}{s}$ frequent items (Cormode and others, 2005). Therefore, by returning the k most occurring items, where $k = \frac{1}{s}$, we are guaranteed to have all items $o_j \in F$. However, it is not the case that all reported items are frequent. That is, there may also be a large number of items $o_j \notin F$ reported.

1.3 Goals

The goal of this thesis is to study prior solutions and propose a new monitoring system for reporting the frequent item set over the union of multiple distributed data streams. If more recent item occurrences are to be weighed more than older occurrences, extensions must be proposed to allow for reporting only recently frequent items. In both cases the system must provide results with exact precision using minimum communication overhead.

In many practical data stream applications, memory requirements are also a major concern. As a result, appropriate memory management policies must be proposed which can be easily integrated into all nodes within the monitoring system. In this case the system may only be able to provide approximate results. We must ensure that these approximations reach a high degree of quality with appropriate empirical analysis.

The final goal of this thesis is to provide a preliminary analysis into the applicability of frequent item monitoring at detecting DDoS attacks. This would include the analysis of the proposed system against a variety of different attack patterns.

Comparisons against other known detection methods will be used to serve as a benchmark to measure the detection quality of the system.

1.4 Organization of Thesis

Chapter 2 discusses prior solutions for solving the frequent items problem and the similar top-*k* elements problem. In Chapter 3 we introduce our approach for monitoring frequent items over distributed data streams. We evaluate our method in Chapter 4 based on a series of defined criteria. Chapter 5 examines the applicability of our method for detecting DDoS attacks. Finally, Chapter 6 gives closing remarks and future directions.

CHAPTER 2

LITERATURE REVIEW

2.1 Single Stream Approaches

Most prior work on monitoring frequent items focuses on creating stream summary data structures. These data structures require a fixed amount of memory but sacrifice correctness in the results. That is, the majority of these solutions address the ε-deficient frequent items problem. The summary data structures proposed in prior work can be broken up into two categories:

1. Counter-based Techniques
2. Time-Sensitive Techniques

2.1.1 Counter-based Techniques

Counter-based techniques work by maintaining a subset of counters smaller than |U|. These methods keep the monitored subset of items small by using various techniques depending on the algorithm (Metwally, Agrawal, and Abbadi, 2005). Each technique has the common characteristic of bounding memory required and providing strong guarantees on approximations based upon a user define error parameter and thus address the ε-deficient frequent items problem directly.

One of the earliest known ε-approximate counting techniques was created by Misa and Gries (1982). Their algorithm required $O(\frac{1}{\varepsilon})$ space and O(1) amortized processing time per update tuple. The same algorithm was rediscovered independently by Demaine *et al.* (2002) and Karp *et al.* (2003). They improved upon the algorithm, allowing O(1) worst case processing time per element by arranging counters in sorted order using a differential encoding.

The original Misa-Gries (MG) algorithm did not specify a user defined error tolerance. Rather, the algorithm only handled the special case where the error tolerance was equal to the support. Recent work has shown, however, that that the algorithm can be adapted to handle general error tolerance values (Lee and Ting, 2006). With these extensions we enhance the pseudocode representation of the MG algorithm provided in (Karp, Shenker, and Papadimitriou, 2003) as shown below.

```
Let $x_1$, $x_2$, …, $x_N$ be a stream of items.
Let Counters be a list of integers indexed by an item $x_i$.
For i = 1 to N do
      If $x_i$ is in Counters Then
            Counters[$x_i$] = Counters[$x_i$] + 1
      Else
            Create a counter for $x_i$, and set Counters[$x_i$] = 1

      If |Counters| > 1 / ε
            For each c in Counters
                  Counters[c] = Counters[c] - 1
                  If Counter[c] = 0 Then Remove Counter[c]
```

FIGURE 2.1 – The MG Algorithm

Essentially the algorithm maintains a set of counters for each item, decrementing $\frac{1}{\varepsilon} + 1$ counters when there are more than $\frac{1}{\varepsilon}$ counters in memory. Since there are N items in the data stream, we can decrement the counters by at most $N/(\frac{1}{\varepsilon} + 1) < \varepsilon \cdot N$ times and hence all counters have error of at most $\varepsilon \cdot N$ (Lee and Ting, 2006). The traditional approach of reporting all frequent items which are under counted, is to report all items with frequency greater than or equal to $(s - \varepsilon) \cdot N$.

Around the same time the MG algorithm was being re-discovered two additional counting techniques were proposed by Manku and Motwani (2002). The first one, called Sticky Sampling is a probabilistic ε-approximate counting technique which can provide frequency counts under counted by at most $\varepsilon \cdot N$ with probability of $1 - \delta$. Their method works by dividing the data stream up into segments each with an associated non-decreasing sampling rate. When a new item is observed it is added to the list of counters based on this sampling rate. At the end of each segment, when the sample rate changes, for each counter we toss a coin decrementing its value by one for each unsuccessful toss. If at any time the value of the counter becomes zero we remove it from memory.

In the same paper Manku and Motwani (2002) proposed a second method, which is the more popular of the two, called Lossy Counting. Lossy Counting is a deterministic ε-approximate counting technique which requires $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon \cdot N))$ space, but in practice performs better on skewed data than both the MG algorithm (Arasu and Manku, 2004) and Sticky Sampling (Manku and Motwani, 2002). Their method works by dividing the data stream up into r rounds of width $\frac{1}{\varepsilon}$. When a new item is observed in a round it is added to the list of counters, and given the value of the previous round $r - 1$. At the beginning of each round, counters with frequency less than or equal to *r* are removed from memory.

More recently a new counting technique has been proposed by Metwally *et al.* (2005) called Space-Saving. This technique requires at most $\frac{1}{\varepsilon}$ counters at any given time. Their method works by storing each counter in a list sorted in order. For each new item observed a counter is added until there becomes $\frac{1}{\varepsilon}$ counters in the list. After this point is reached, each remaining new counter replaces the counter of the least occurring item currently in the list. By managing new counters in this way, frequency counts are over counted by at most $\varepsilon \cdot N$. Recently, work by Liu *et al.* (2006) enhanced this approach by identifying all counters with frequency less than $\varepsilon \cdot N$ and replacing the oldest counter. The main contribution of their work was the creation of a method in which to store time information while using little additional space. They did this by devising a method called fractionization, where they added the inverse sum of the natural log of each timestamp for an item to its frequency count. They thus defined the oldest counter as the counter with the smallest sum of timestamps.

2.1.2 Time-Sensitive Counting Techniques

Most recent work on time-sensitive frequency counts utilizes the concept of a sliding window model. Sliding windows consist of the last *e* item occurrences observed in a data stream. The two most common types define *e* as either a fixed value (fixed-sized sliding window) or as variable (variable-sized sliding windows). Each of these two fundamental types can additionally be extended to represent a variety of other sliding window models, including time-based sliding windows (Arasu and Manku, 2004).

The first deterministic ε-approximate counting technique for finding frequent items over sliding windows was proposed by Arasu and Manku (2004). Their algorithm required $O(\frac{1}{\varepsilon} \cdot \log^2 \frac{1}{\varepsilon})$ space and $O(\frac{1}{\varepsilon} \cdot \log \frac{1}{\varepsilon})$ processing time per update tuple for a fixed-size sliding window. The algorithm works by creating a series of copies of the data stream, each broken up into segments. The size of each segment varies between the copies but segment sizes remain uniform within. Each individual segment is constructed by running an instance of the MG algorithm. Frequency counts of items are determined by examining a selective number of these segments in the different copies, which are management carefully to guarantee approximate results.

More recently Lee and Ting (2006) have proposed a new method which is more efficient. Their method requires only $O(\frac{1}{\varepsilon})$ space and processing time per update tuple for representing fixed-size sliding windows. Additionally, they expanded their method using a similar approach by Arasu and Manku (2004) to handle variable-sized sliding windows using $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon \cdot N))$ space and processing time.

Unlike the previously described algorithm, their method uses the MG algorithm directly. To maintain frequency counts over a sliding window they replaced each counter of the MG algorithm with a special λ-counter. Briefly a λ-counter represents a data stream as a sequence of bits each with a labeled position starting at position one. For each position that the item represented by the counter occurs, a 1-bit is recorded at that position. These positions are grouped together into λ sized blocks indexed sequentially starting at index one. A block is defined as significant if it falls in or overlaps with the current sliding window and contains the $(i\lambda)$th 1-bit (or the $(i\lambda)$th occurrence of the item) where $i \geq 1$. The value of the λ-counter is thus defined as $\lambda \cdot |Q| + l$ where Q is the list of significant blocks and l is the number of ones (or occurrences) observed since the last significant block. For example in Figure 2.2, if we define $\lambda = 2$ and the size of the window as 10, the list of significant blocks Q = {3, 5}. The resulting value of the λ-counter is thus equal to five.

position	1	2	3	4	5	6	7	8	9	10	11	12
bit stream	1	1	1	0	0	1	1	0	1	0	1	0
block	1		2		3		4		5		6	

FIGURE 2.2 – Example problem using λ-counter

2.2 Centralized Monitoring Approaches

Many monitoring systems process multiple distributed data streams by forwarding update tuples to a centralized node (Babcock and others, 2002). At this node each received update tuple can be modeled as only a single data stream in a straightforward fashion. This is beneficial, since a large number of single stream frequent item monitoring methods have been proposed (see Section 2.1). Most of these methods are optimized to require minimal space and processing time. Despite these optimizations, however, the response time of a centralized approach may still be long. It is likely that the merged single stream will be very large, thus requiring a large amount of data to be processed. Additionally, forwarding each data stream will result in heavy communication congestion and in some cases it is not feasible (Aggarwal and others, 2006). For example, examine the case where the data streams are being collected by distributed wireless sensors. In this environment the power consumption for each sensor is dominated by the amount communication involved (Madden and others, 2003). Thus forwarding the data streams will likely result in low battery life for each sensor.

2.3 Distributed Monitoring Approaches

To alleviate some of the short comings of the centralized monitoring approach, a variety of distributed solutions have been developed. All these solutions have two important characteristics which make them more efficient. First each distributed monitoring node conducts some of the processing locally. Second communication is only conducted when certain criteria is reached or when results are requested. Additionally, when communication does occur, only a concise summary of the local data stream is transmitted. These two characteristics allows distributed monitoring approaches to both reduce the amount of communication needed and decrease the processing time by utilizing distributed resources.

Prior work on finding frequent items using the distributed approach can be broken into two categories. The first category consists of methods defined as one-time query approaches. One-time query approaches only provide results once over a point in time (Babcock and others, 2002). These methods are not designed for continuous monitoring, but can be repeated within short time intervals to simulate results (Babcock and Olston, 2003). The second category consists of methods defined as continuous query approaches.

Continuous query approaches provide at any time, the results equal to the output of a one-time query approach (Golab and Ozsu, 2005).

2.3.1 One-Time Distributed Approaches

The recent one-time query approach by Manjhi *et al.* (2005) addressed the problem of determining time-sensitive (recent) frequent items over distributed data sources. They did this by installing an ε-approximate counting technique at each monitoring node, and transmitting local frequency counts every T time units to a centralized node. Upon receiving the local counts, they are combined with previously received counts. More emphasis is placed on recent item occurrences by deprecating previously received counts in an exponentially decaying fashion.

The authors realized that transmitting all frequency counts to a centralized node would result in excessive communication cost. Additionally, if there are a large number of monitoring nodes, the centralized node will be overwhelmed by the amount of data received. To address this issue, they proposed using a multi-level hierarchical communication architecture to reduce the load on the centralized node. A multi-level hierarchical communication architecture is very similar to the single-level architecture introduced in Section 1.2.1. The only difference is the introduction of intermediate nodes between the monitoring nodes (defined as leaf nodes) and the centralized coordinator node (defined as the root node) (Cormode and Garofalakis, 2005b). In Figure 2.3 we give an example of a three-level hierarchical communication structure.

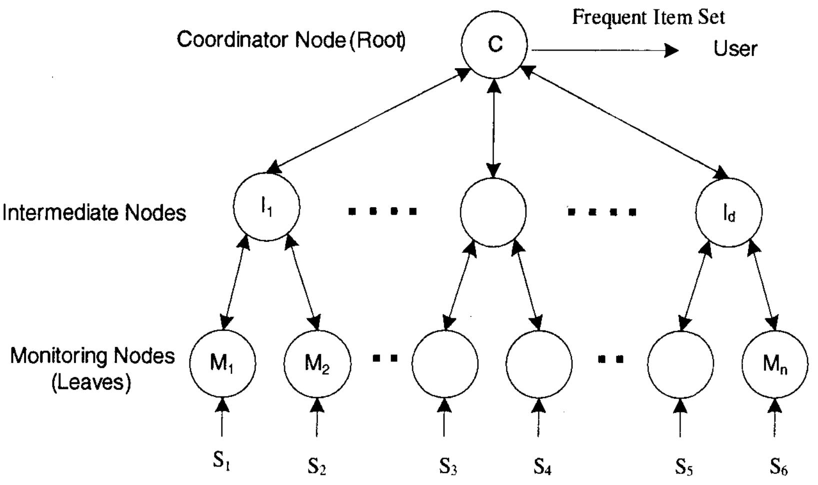


FIGURE 2.3 – Multi-level hierarchical communication architecture.

The intermediate nodes were used by the authors to additively combine the frequency counts received from their child nodes. Additionally, they introduced the concept of a precision gradient to reduce communication load on any single link. Roughly speaking, the ε-approximate counting techniques installed on each leaf node (monitoring node) undercounts the true frequency of an item. Frequencies of value zero do not need to be transmitted. Instead of introducing the maximum amount of error at the leaf nodes, they varied the degree of error tolerance at each level. The authors provided strategies for setting the error tolerance at each level, and demonstrated that their strategies worked well for reducing both load on the coordinator node and reducing load on any single communication link.

Several approaches for solving the similar problem of finding the top-*k* items in the distributed data stream environment have been introduced. All these methods have shown to successfully reduce communication cost. One related group of algorithms is the threshold based algorithms described by Yu *et al.* (2005) in their recent work. Each of the algorithms described (TPUT, TPAT, TPOR, and HPT) requires three steps which are generalized as follows:

1. Step 1: The local top-*k* list of each monitoring node is collected at a centralized node.
2. Step 2: The information gathered at the centralized node is used to assign each monitoring node a threshold or supplies it with information needed to create a threshold. Each monitoring node responds with a list of items, each with frequency greater than the given threshold. The response is then used to create a top-*k* candidate set.
3. The exact frequencies are collected for each item in the candidate set and the top-*k* items are reported.

The goal of each of these steps is to gather successively more information about the items observed in the network to reach a final consensus on the global top-*k* set. This is in contrast with simply forwarding all item frequencies to a centralized node. Ideally the threshold based approaches will only gather information on a subset of items, thus reducing the amount of communicated when compared to the simpler approach.

2.3.2 Continuous Distributed Approaches

The most recent work to address the issue of continuous monitoring over distributed data sources was the work of Cormode *et al.* (2005) which was later expanded. The expanded approach required O($\frac{1}{\varepsilon}^2 \cdot \log \frac{1}{\delta}$) space and O(log $\frac{1}{\delta}$) time per update tuple, where ε is a user define error tolerance and δ is a probability of failure (Cormode and Garofalakis, 2005a). Additionally, their analysis showed that the worst case communication cost for their method was comparable with that of periodic one-time approaches.

Both the methods in (Cormode and others, 2005) and in (Cormode and Garofalakis, 2005a) work by maintaining at each monitoring node a summary of the observed local data stream and a corresponding predication model. Throughout the monitoring process each monitoring node compares its local summary with its corresponding prediction model. If the prediction model deviates from the actual local summary by more than a user defined error tolerance, the local summary (and possibly a new predication model) is communicated to a centralized node. Thus if the prediction model approximates the actual data stream effectively, no communication is needed.

Since the centralized node at all times contains the approximate frequency counts of all items, this method is able to answer a variety of different queries. Included is the ability to monitor top-k items and provide a list of heavy hitters. Heavy hitters in this case is not the same as the ε-deficient frequent items problem. Since we only have the approximate frequency counts, we return all items which are above $s \cdot \check{N}$, where $\check{N}$ is the sum of all approximate frequencies which estimates N.

Previously, Babcock and Olston (2003) addressed explicitly the problem of monitoring the top-k items in the distributed environment. The basic concept for their method was to maintain at each monitoring node the global top-k set locally. Conceptually, this was done by shifting the frequencies of items amongst the monitoring nodes. This shifting was done in the fashion so that the adjusted frequency of each item in the global top-*k* set was greater than the adjusted frequency of any item not in the global top-*k* set. As long as this arrangement of adjusted frequencies remained, the validity of the global top-*k* set sat unchanged.

In the event that a monitoring node can no longer maintain the global top-*k* set, a process called resolution was initiated. The process of resolution may result in one of two outcomes:

1. The coordinator determines that the global top-*k* set has remained unchanged by using a subset of frequency counts from the invalidated monitoring node and locally stored frequency counts gathered during the process of shifting item frequencies. In this scenario communication is only required amongst the coordinator node and the invalidated monitoring node.
2. The coordinator calculates the new global top-*k* set by gathering a subset of frequency counts from each monitoring node. Frequencies are then shifted once more so that the new global top-*k* set is now maintained on each monitoring node. In this scenario communication is conducted amongst the coordinator node and all monitoring nodes.

Additionally, Babcock and Olston (2003) introduced another method which they called the caching approach. This method worked in a similar fashion as the previously discussed algorithm proposed by Cormode and Garofalakis (2005a). The caching approach required that each monitoring node send the frequency of an item with value greater than $\frac{\varepsilon}{m}$ its previously sent frequency to the coordinator node, where ε is a user defined error tolerance and m is the number of monitoring nodes.

CHAPTER 3

FREQUENT ITEM MONITORING

3.1 Overview

In this chapter we propose a new monitoring approach for solving the frequent items problem in the distributed data stream environment. Our method is a direct modification to the top-*k* monitoring approach by Babcock and Olston (2003), in which will we call Top-K Monitoring. Briefly, instead of maintaining the global top-*k* set locally on each monitoring node we will maintain the global frequent item set.

In Section 3.2 we will introduce, in detail, the steps required to maintain the global frequent item set locally on each monitoring node. In this section, we will assume that the goal is to monitor the exact frequent item set. To provide exact answers we must store a frequency count for each unique item observed in the data stream.

In Section 3.3 we will introduce the modifications needed to provide the approximate frequent item set. The goal in this section will be too reduce memory requirements by utilizing the summary data structures introduced in Section 2.1. Although Top-K Monitoring provided approximate results for the top-*k* elements problem, they did not utilize these prior summary structures. As a result, their monitoring approach still required a counter for each unique item observed despite introducing approximate results.

Finally in Section 3.4 we will examine ways to solve the recently frequent items problem. That is, we will examine ways to weigh recent item occurrences more than older ones. In the same fashion as the previous sections, we will begin by first introducing ways to provide the exact recently frequent item set and then introduce methods to provide approximate results with the goal of reducing memory required.

3.2 Exact Frequent Item Monitoring

The frequent item monitoring approach begins with an initialization phase. There are three ways to accomplish this task. One option is to issue an efficient one-time distributed frequent item query (see Section 2.3.1). The advantage of this approach is that communication is only conducted after the initialization time period has ended. Additionally, when communication does occur, only a concise summary of the initially observed data stream will be transmitted. Although initial communication cost is minimal with this approach, the same causes for this are also its downfall. That is, the drawback of this approach is that the monitoring process will not begin until the initialization time period has passed. It is also not clear how long of an initialization period is needed. If the initialization period is too short, there may not be a significant reduction in communication when compared to other approaches.

The second initialization option is to forward all update tuples to the coordinator node. This method is representative of a centralized monitoring approach (see Section 2.2). Since all update tuples will be forward to a centralized node, initial communication cost may be high. However, the monitoring process will be able to begin immediately, since the centralized coordinator node has full access to the frequency of all items.

Finally, the third option is to simply begin our monitoring method immediately. Once the system starts, if no items have been observed their frequencies will be zero. That is, the set of frequent items is empty and does not need to computed. This approach simplifies the initialization phase since we do not need to design a specialized method. Additionally, we do not need to be concerned with setting an initialization time period.

Once the initialization phase is completed, the coordinator node sends to each monitor the current frequent item set. Similar to Top-K Monitoring, we maintain the global frequent item set locally on each monitor. To do this we have designed a series of parameterized constraints (or local requirements) which are installed on each monitoring node. These constraints are used to detect if the global frequent item set has changed over time, and consists of two core components.

The first component of the parameterized constraints is a local threshold value T_i, kept by each corresponding monitoring node N_i. Initially $T_i = 0$ and for each item occurrence observed at node N_i, including any observed during the initialization phase, we increment T_i by the user defined support value s. By incrementing the threshold value in this fashion it is clear that $T_i = s \cdot |S_i|$, where $|S_i|$ is the number of update tuples in the locally observed data stream, or in other words, the total number of locally observed item occurrences. If we sum the threshold values for each monitoring node we see that $T = \sum_{1 \le i \le m} T_i = \sum_{1 \le i \le m} s \cdot |S_i| = s \cdot N$, since by definition N is the total number of item occurrences observed across all nodes.

The second component of the parameterized constraints is a series of adjustment factors. The notation for these adjustment factors are borrowed from the Top-K Monitoring method and are used in a similar fashion (Babcock and Olston, 2003). For each item o_j and node N_i we define a corresponding adjustment factor $\delta_{j,i}$. For the correctness of our monitoring method, we require that each adjustment factor meet three requirements:

1. For each item o_j, its corresponding adjustment factors must sum to zero across all nodes: $\sum_{0 \le i \le m} \delta_{j,i} = 0$.

2. For each item $o_F \in F$, its corresponding adjustment factor stored at the coordinator node is greater than or equal to zero: $\delta_{F,0} \ge 0$.

3. For each item $o_F \notin F$, its corresponding adjustment factor stored at the coordinator node is less than or equal to zero: $\delta_{F,0} \le 0$.

With the two core components of the parameterized constraints described, we can now show how they are defined and how they are used to determine if the validity of the global frequent item set has changed. For each item observed at monitoring node N_i, the following constraints are installed:

1. If an item $o_j \in F$, then the installed constraint is defined: $c_{j,i} + \delta_{j,i} \ge T_i$, where $c_{j,i}$ is the frequency count of o_j and $\delta_{j,i}$ is the adjustment factor corresponding to item o_j.

2. If an item $o_j \notin F$, then the installed constraint is defined, $c_{j,i} + \delta_{j,i} < T_i$.

If all the parameterized constraints hold for each node, then for every $o_j \in F$, $\sum_{1 \le i \le m} c_{j,i} + \sum_{0 \le i \le m} \delta_{j,i} \ge \sum_{1 \le i \le m} T_i$ or $\sum_{1 \le i \le m} c_{j,i} \ge s \cdot N$. Likewise for every $o_j \notin F$, $\sum_{1 \le i \le m} c_{j,i} + \sum_{0 \le i \le m} \delta_{j,i} < \sum_{1 \le i \le m} T_i$ or $\sum_{1 \le i \le m} c_{j,i} < s \cdot N$.

We see now that our monitoring method works by dividing the global threshold $s \cdot N$, which determines which items are frequent, amongst each monitoring node. We then shift item occurrences amongst the nodes in the form of adjustment factors. As long as the adjusted frequency of each frequent item o_j ($c_{j,i} + \delta_{j,i}$) is above the local threshold (T_i), or if infrequent below the threshold, on each monitoring node then validity of the global frequent item set holds. At any time in which a constraint is broken, a process called resolution is initiated.

3.2.1 Resolution

Whenever a local parameterized constraint is broken on any monitor node a three phase process called resolution is initiated. The purpose of this process is two-fold. First, the validity of the global frequent item set is checked. If the global frequent item set has changed, we must notify all monitoring nodes with the list of new frequent items and the list of all items no longer frequent. Second, at the conclusion of the process, item frequencies are shifted in the form of adjustment factors so that all local constraints become valid. The resolution process we use is a modification to Top-K Monitoring. All modifications made are described below.

To begin the resolution process, in Phase 1, the monitoring node containing the invalid constraint N_I sends a message to the coordinator. The contents of this message include: the frequency counts, corresponding adjustment factors, and item identifiers of all items involved in an invalid constraint. Once the information regarding each invalid constraint is sent to the coordinator, the local threshold value T_i is transmitted. Note that information regarding the frequencies of every item which is a member of the frequent item set is not needed. The membership of any item in F is independent of any other item. This property is distinctive of our monitoring approach. In Top-K Monitoring, when any constraint is broken, information regarding all items with broken constraints and the global top-k items are transmitted to the coordinator. As a result, the message size is considerably large when k is assigned a large value. As we will see later, the reduced format of our messages will play an important part in reducing overall communication cost.

Once all relevant information is gathered at the coordinator node the second phase can begin. In Phase 2, the coordinator attempts to determine if the global frequent item set remains valid. Recall in our definition of adjustment factors, for each item o_j the coordinator node may have an corresponding adjustment factor $\delta_{j,0}$ stored locally. As a result, the coordinator may be able to sacrifice a portion of it's locally store adjustment factors to node N_I in order to restore invalid constraints. In order to ensure if this is possible, for each violated constraint the coordinator performs the following validation tests:

1. If $o_j \in F$ then the test performed is $c_{j,I} + \delta_{j,I} + \delta_{j,0} \geq T_I$.
2. If $o_j \notin F$ then the test performed is $c_{j,I} + \delta_{j,I} + \delta_{j,0} < T_I$.

If all validation tests pass, then a process called reallocation is initiated (see Section 3.2.2) and resolution terminates at Phase 2. All interactions involved from initiation of resolution to termination after resolution is depicted in Figure 3.1. If any one test fails during validation testing, however, we are not able to determine if the global frequent item set has changed. At this point more information is needed and the resolution process continues.

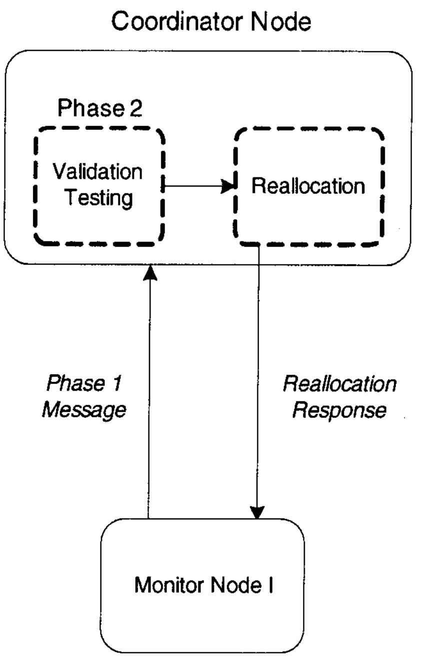


FIGURE 3.1 – Reallocation termination after Phase 2.

In the final phase of resolution, Phase 3, the coordinator must contact all remaining monitoring nodes. That is, for each node $N_i : i \neq I$, the coordinator node must request information regarding each item involved in an invalid constraint. Response messages in this phase, is of the same format as that received from node N_I in Phase 1. With complete knowledge on the global frequency of each item involved and the global threshold, the coordinator node can now determine if the global frequent item set has changed directly. Notice that Phase 3 can be labeled as a (re)synchronization phase, since all monitors in the network are contacted and given the new frequent item set (shown in Figure 3.2). To conclude this final phase, the coordinator initiates the reallocation process.

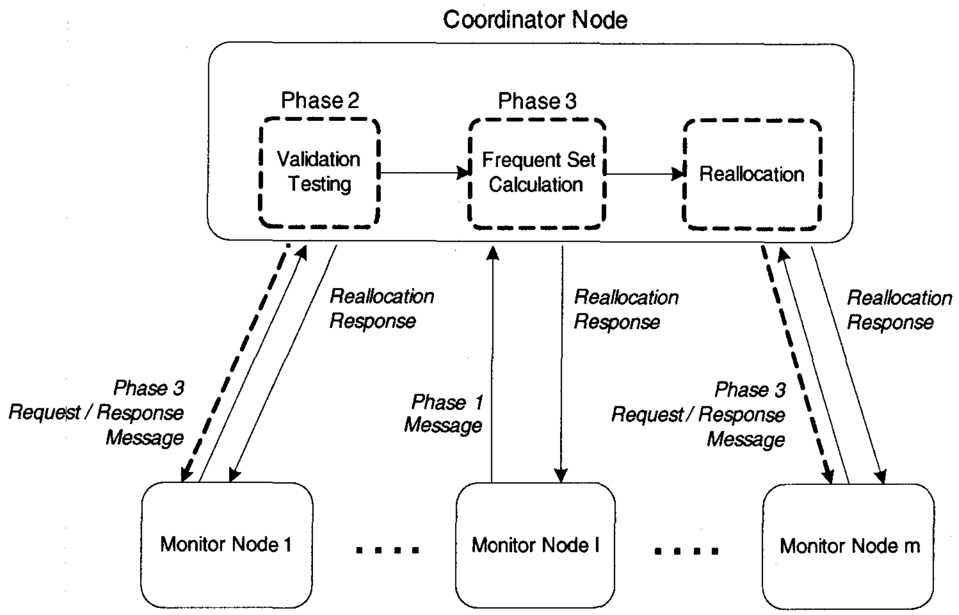


FIGURE 3.2 – Reallocation termination after Phase 3.

3.2.2 Reallocation

Before the resolution process can terminate, adjustment factors must be reassigned in such as fashion that all constraints are satisfied for the current global frequent item set. This requires that item occurrences must be rearranged amongst all the nodes involved in the resolution process. Borrowing from Top-K Monitoring, we call these set of nodes η the participating nodes. If resolution terminated after Phase 2, then $\eta = \{N_I, N_0\}$, otherwise, $\eta = \{N_0, N_1, \ldots, N_m\}$ (Babcock and Olston, 2003). The process responsible for these reassignments is called reallocation. Like resolution, this process is a modification of the same process found in Top-K Monitoring. The process and changes made are now described.

The first step of reallocation is the summation process of all available information. That is, we sum the value of each available adjusted frequency (frequency plus the adjustment factor) for a given item involved in an invalid constraint.

Additionally, we sum all threshold values received from monitoring nodes in η. If reallocation was initiated by Phase 3 of resolution, then the summation process will give us the global frequency for each item and the global threshold. On the other hand, if it was Phase 2 that initiated reallocation, then we only have the partial frequencies of each item and the threshold of N_I. Since only adjustment factors are stored at the coordinator, the partial frequencies in this case will be equal to $c_{j,I} + \delta_{j,I} + \delta_{j,0}$, for a given item o_j. Once summation is completed, we calculate the distance Δ_j of each aggregated frequency from the aggregated threshold.

The second step of reallocation is the "tightening" process. For each monitoring node $N_i \in \eta$ and item o_j, we assign a new adjustment factor $\delta'_{j,i}$ so that that the adjusted frequency is equal to the local threshold value. Doing this step alone is enough to guarantee that each item in the global frequent item set will have valid constraints, since the adjusted frequency $c_{j,i} + \delta'_{j,i} = T_i$, for any given item o_j and monitoring node $N_i \in \eta$ involved in the process.

The final and third step, assigns a portion of Δ_j to each new adjustment factor assigned in the previous step. The amount added is based on an allocation parameter $0 \leq F_i < 1$ corresponding to each node N_i. Allocation parameters are defined in such a fashion to control the amount of Δ_j allocated to each node and is required that $\Sigma_{0 \leq i \leq m} F_i = 1$. This notation is similar to that of Top-K Monitoring with the exception that $F_0 \neq 1$. That is, we can not assign Δ_j entirely to the coordinator node. Any item $o_j \notin F$ must have a value less than its local threshold. As a result of this requirement and the assignments made in the previous step, we must therefore assign a portion Δ_j to the new adjustment factors in order for all constraints to be valid.

Given the description above, the reallocation procedure can be expressed formally with only two expressions.

1. $\Delta_j = \sum_{i\in\eta} c_{j,i} + \sum_{i\in\eta} \delta_{j,i} - \sum_{i\in\eta} T_i$
2. $\delta'_{j,i} = T_i - c_{j,i} + F_i \cdot \Delta_j$

The first expression represents the summation process, while the second expression represents final steps. For each item o_j involved in a violated constraint and node in η, both expressions are evaluated to determine the new adjustment factor $\delta'_{j,i}$ where $i \in \eta$ represents node N_i. Comparing these two equations to those used in Top-K Monitoring will show that the reallocation method originally designed can be re-used. Assigning the parameters used in Top-K Monitoring appropriately will result in the definitions given above.

3.3 Approximate Frequent Item Monitoring

In the previous section we introduced a modification to Top-K Monitoring for continuously tracking the exact global frequent items. In order to allow exact solutions, counters for each unique item observed must be kept. The value of each counter must represent the true frequency of its corresponding item at all times. If the number of unique items observed in the data stream is large, this will result in impractical memory requirements. For example, consider the problem of tracking frequent users to a website based off the user's IP address. With the new IP version 6 addressing scheme it is possible to have up to 2^{128} or $3.4 \cdot 10^{38}$ unique IP addresses, and thus a large number of corresponding counters (Hinden and Deering, 2006).

To reduce and bound memory requirements for our monitoring approach, we reviewed a number of summary data structures (See Section 2.1.1). Each of these summary structures reduces memory requirements at the cost of providing approximate frequency counts. If we can integrate one of these structures into our monitoring nodes, then this would allow us to bound and reduce memory requirements across the network. However, integrating these structures into our monitoring nodes will only allow our method to provide approximate results or the ε-deficient frequent items.

3.3.1 Summary Structure Selection

To decide which summary data structure to integrate into our monitoring nodes, we referenced the work of Metwally *et al.* (2005). As previously discussed, the authors introduced a new counter-based summary structured called Space-Saving. Also included in their work, however, is an extensive comparison of their proposed method with two other summary data structures. These two methods included the MG algorithm and another summary structure called GroupTest. The results of their comparisons indicated that the MG algorithm consistently ran faster and used five times less space than the Space-Saving method. However, the MG algorithm produced a significantly larger number of false positives. That is, it identified a large number of items as frequent which was not frequent in reality. Of the three techniques analyzed, Space-Saving produced the fewest false positives. Therefore, if the quality of the results is the leading factor in our selection process, Space-Saving is the likely candidate to use to integrate into our monitoring nodes. However, a quick observation of the MG algorithm shows that we can greatly improve upon the quality its results.

Recall, from Section 2.1.1, that in the single stream approach if a new item is observed and there are more than $\frac{1}{\varepsilon}$ counters currently in memory, the MG algorithms proceeds to decrement all counters by one. These decrements account for the error introduced in each frequency count. The maximum number of decrements was determined to be $\varepsilon{\cdot}N$. That is, each frequency count is at most under counted by $\varepsilon{\cdot}N$ (Lee and Ting, 2006). Finally, a frequent item was defined as any item which has frequency greater than or equal to $(s - \varepsilon){\cdot}N$.

Let f define the true frequency of an item which is frequent, we see that if the frequency is undercounted by the maximum amount, then $f - \varepsilon{\cdot}N \geq s{\cdot}N - \varepsilon{\cdot}N$ or simply $f \geq s{\cdot}N$. However, in practice this worst case is observed rarely. That is, it is unlikely any counter is undercounted by $\varepsilon{\cdot}N$, but rather undercounted by a value less than $\varepsilon{\cdot}N$. From this observation we modify the threshold used by the MG algorithm to report all items with frequency greater than or equal $s{\cdot}N$ minus the number of decrements. With this modification in the threshold we will see that the quality of the MG algorithm is now comparable to that of Space-Saving. Additionally, we saw that the MG algorithm out-performed Space-Saving in all other parameters. Thus, we selected this counting technique to integrate into our monitoring nodes.

The integration of the MG algorithm into our monitoring nodes requires two steps. The first step requires that we now maintain the frequency counts in the matter required by the MG algorithm. The second step is that we must redefine the local threshold values. For monitoring node N_i we now define the local threshold T_i as the number of item occurrences observed locally, minus the number of decrements introduced locally. More formally, $T_i = s \cdot |S_i| - d_i$ where d_i is the number of local decrements. Similar to our previous analysis, if we sum the threshold values for each monitoring node we see that $T = \sum_{1 \le i \le m} T_i = \sum_{1 \le i \le m} s \cdot |S_i| - d_i = s \cdot N - D$, where D represents the total number of decrements observed which can be at most $\varepsilon \cdot N$.

3.3.2 Approximation Bounds

Equally important to reducing and bounding memory requirements, is to provide guarantees on the approximations made on each frequency count. In the single stream approach the MG algorithm under counts any frequency by at most $\varepsilon \cdot N$ (Lee and Ting, 2006). We will now show that equal guarantees can be made by integrating the MG algorithm into each monitoring node.

THEOREM 1. Let each monitoring node maintain local frequency counts using the MG algorithm. For any given item o_j, it global frequency $c_{j,\bullet}$ is at most undercounted by $\varepsilon \cdot N$.

PROOF. Each monitoring node uses the MG algorithm to summarize a single stream of size $|S_i|$. Substituting N with $|S_i|$, each approximate frequency count $\hat{c}_{j,i}$ at all times $c_{j,i} - \varepsilon \cdot |S_i| \le \hat{c}_{j,i} \le c_{j,i}$. Summing across all monitoring nodes, to get the global frequency, thus yields $\sum_{1 \le i \le m} (c_{j,i} - \varepsilon \cdot |S_i|) = \sum_{1 \le i \le m} c_{j,i} - \sum_{1 \le i \le m} \varepsilon \cdot |S_i| \le \sum_{1 \le i \le m} \hat{c}_{j,i} \le \sum_{1 \le i \le m} c_{j,i}$. Since by definition $\sum_{1 \le i \le m} |S_i| = N$, we see that $c_{j,\bullet} - \varepsilon \cdot N \le \hat{c}_{j,\bullet} \le c_{j,\bullet}$. Thus the global frequency of any item is at most undercounted by $\varepsilon \cdot N$. □

3.3.3 Adjustment Invariant Maintenance

The MG algorithm selected to be integrated on each monitoring nodes does not store in memory any counter with frequency equal to zero. Additionally, the method may decrement counters until the frequency becomes zero. However, recall for each counter $c_{j,i}$ there is a corresponding adjustment factor $\delta_{j,i}$. If the corresponding adjustment factor $\delta_{j,i} \neq 0$, we can not remove it from memory. Doing so would result in the sum of all associated adjustment factors to not equal zero. This will clearly invalidate our first adjustment factor requirement, and thus invalidate our entire frequent item monitoring approach. Therefore, before we remove counters from memory appropriate steps must be taken to manage the adjustment factors carefully.

When removing a counter $c_{j,i}$ with frequency of zero, the corresponding adjustment factor $\delta_{j,i}$ may be in one of three states. The first state is when the adjustment factor $\delta_{j,i} = 0$. In this scenario counter $c_{j,i}$ and its corresponding adjustment factor $\delta_{j,i}$ can be removed from memory directly. The counter no longer has any value to the monitoring process, and removing it from memory will not affect the sum of associated adjustment factors. The second state is when the adjustment factor $\delta_{j,i} > 0$. In this scenario we cannot remove the counter from memory, since its adjustment factor still holds value. We therefore continue to store the counter in memory until $\delta_{j,i} = 0$, or until the first state is reached. The third and final state, is when the adjustment factor $\delta_{j,i} < 0$. In this scenario we forward the value of the adjustment factor to the coordinator resulting in the new value of $\delta_{j,i} = 0$, or in other words resulting in the state change to the first state. At this point the counter and associated adjustment factor are removed from memory.

When the coordinator receives an adjustment factor corresponding to item o_j from a monitoring node N_d, we require that it be redistributed to the other monitoring nodes. More precisely, we select a subset of monitoring nodes with corresponding adjustment factor $\delta_{j,i} > 0$. Theorem 2 shows that it is always the case that at least one such monitoring node exists.

THEOREM 2. If there is an adjustment factor $\delta_{j,d} < 0$ with corresponding counter $c_{j,d} = 0$ at node N_d, then there exists a monitoring node N_P containing $\delta_{j,P} > 0$.

PROOF. With adjustment factor requirement 1 we know that $\sum_{0 \le i \le m} \delta_{j,i} = 0$. Thus if there is a $\delta_{j,i} < 0$, there must be a node N_p containing corresponding $\delta_{j,p} > 0$. Since the adjustment factor is $\delta_{j,i} < 0$ and its corresponding counter $c_{j,i} = 0$, we know that the item o_j is globally infrequent. This must be true since we require the adjusted frequency $(c_{j,i} + \delta_{j,i})$ of any globally frequent item to be less than the local threshold $T_i \ge 0$. With adjustment factor requirement 3, we know that $\delta_{j,p}$ is not at the coordinator node. Thus $\delta_{j,p}$ is located on a monitoring node N_p. □

To determine the set of monitoring nodes with adjustment factors $\delta_{j,i} > 0$, we can store all adjustment factor assignments at the coordinator node to prevent polling. With this knowledge the coordinator can both determine which nodes to forward the negative adjustment factor to and how much of its value to forward to each. We only want to forward enough so that to cancel the positive adjustment factor out at the receiving nodes. Making the receiving nodes adjustment factor negative will require us to forward to the coordinator once again at a later time. This would result in wasted communication cost.

Finally, the coordinator itself may also contain an associated adjustment factor. This adjustment factor will be negative since the item in question is globally infrequent.

As a result, whenever the coordinator is assigning new adjustment factors, if its determined that there are no longer any monitoring nodes containing a negative adjustment factor, the coordinator must redistribute it owns negative adjustment factor. This may occur upon receiving a forwarded adjustment (see Appendix II for process description) or during the reallocation process. In both cases the coordinator node must also include its own negative adjustment factor in each new assignment made.

The overall purpose of the described adjustment factor maintenance policy, is to remove all adjustment factors corresponding to item o_j with global frequency $\sum_{1 \le i \le m} c_{j,i} = 0$. In this scenario there is no need to maintain the frequency count of such an item on any monitoring node. Therefore, we take advantage of adjustment factor requirement 1, to remove any remaining positive adjustment factors.

3.3.4 Memory Requirements

With the method for maintaining both the frequency counts and their corresponding adjustment factors we can now determine the memory requirements for both the monitoring nodes and the coordinator.

THEOREM 3. Each monitoring node uses at most $O(\frac{m}{\varepsilon})$ counters and corresponding adjustment factors.

PROOF. Using the MG algorithm and the adjustment factor maintenance policy given, we know that there are at most $\frac{1}{\varepsilon}$ plus any positive adjustment factors. In the worst case each item observed locally on each node is unique globally and requires an associated adjustment factor. In this case each monitoring node will have $\frac{(m-1)}{\varepsilon}$ positive adjustment factors plus its own local $\frac{1}{\varepsilon}$ counters. Adding these two totals together, we see that the memory requirements is O($\frac{m}{\varepsilon}$). □

Since the coordinator only stores adjustment factors we can also bound the memory requirements.

THEOREM 4. The coordinator node has at most O($\frac{m^2}{\varepsilon}$) adjustment factors stored in memory.

PROOF. The coordinator node maintains the adjustment factor assignments made to each node. In the worst case each item observed locally on each node is unique globally and requires an associated adjustment factor. If this is the case, we have $\frac{m}{\varepsilon}$ unique items in the system and $\frac{m}{\varepsilon} \cdot (m + 1)$ adjustment factor assignments. Thus the coordinator node stores at most O($\frac{m^2}{\varepsilon}$) adjustment factors in memory. □

3.4 Recently Frequent Item Monitoring

To weigh new item occurrences more than older ones, we can adopt directly the sliding window model. We select this model since it is the most often used in practice (Zhu and Shasha, 2003). The concept of sliding windows was introduced in Section 2.1.2. How sliding windows can be integrated into our monitoring approach is now discussed.

For monitoring exact recently frequent items, item occurrences must be buffered in memory. If we are using a time-based sliding window (an example of a variable sized sliding window), item occurrences within the last t time units must be buffered (Arasu and Manku, 2004). On the other hand, if we are using a count-based sliding window (an example of a fixed size sliding window) we must buffer the last *e* item occurrences in the data stream (Demaine, Lopez-Ortiz, and Munro, 2002). Along with the buffered data stream, we must still maintain frequency counts for each observed item. Using either sliding window types, item frequency counts may be decremented to zero and have an associated adjustment factor. To prevent adjustment factors from accumulating and to free additional space for the window buffer, we can manage adjustment factors in the same fashion as described in Section 3.3.3.

If the observed data stream is very large, buffering it entirely into memory may not be practical. As a result, approximate solutions can be given using the time-sensitive counting techniques introduced in Section 2.1.2. Additionally, the most recent time-sensitive counting technique, introduced by Lee and Ting (2006), uses the MG algorithm directly. Integrating this method into our monitoring method requires the same steps introduced previously in this work.

CHAPTER 4

EXPIREMENTAL EVALUATION

4.1 Data Sets and Monitoring Description

We implemented our frequent item monitoring approach in JAVA to simulate the behavior of the proposed method. All interactions between the monitoring nodes and the coordinator node were simulated on a single machine. Various statistics were gathered on each simulation run to measure the performance of our method. The descriptions of some of these statistics are given in Section 4.2.

The data sets used to represent the distributed data streams was acquired from the Internet Traffic Archive. The Internet Traffic Archive is a data set repository used to provide network traces to study network dynamics, usage characteristics, and growth patterns (ita.ee.lbl.gov, 2000). The data sets publicly available include: LAN and WAN packet traces, HTTP logs from web servers, and raw internet routing data.

To evaluate our modifications to Top-K Monitoring for tracking frequent items, we selected two data sets. The first data set consists of wide-area network traffic between Lawrence Berkeley Laboratory and the rest of the world (Paxson and Floyd, 1995). The data set was created using tcpdump, capturing approximately 1.8 million TCP packets. Information gathered on each packet include: timestamp, source addresses, destination address, source TCP port, destination TCP port, and number of bytes communicated. All addresses were renumbered using a collection of scripts to preserve the privacy of the network users. A few example packet records collected are given in Table I below.

TABLE I

LAWRENCE BERKELEY LABORATORY EXAMPLE DATA

Timestamp (μs)	Source Address	Destination Address	Source TCP Port	Destination TCP Port	Bytes Sent
0.010445	2	1	2436	23	2
0.023775	1	2	23	2436	2
0.026558	2	1	2436	23	1
0.029002	3	4	3930	119	42
...	...	...	...	...	...
7199.999857	399	138	1663	23	0

For our simulation experiments we monitored frequent users (source addresses). As a result, we did not need all six fields and created a new text file containing only the timestamps and source addresses. Additionally, the TCP packets for this data set were collected on a single node. To simulate a distributed environment we evenly assigned the records amongst four monitoring nodes.

The second data set used in our evaluation, consisted of the 1998 World Cup web site logs (Arlitt and Jin, 1998). The logs were gathered between April 26 and July 26, 1998 containing web requests made to each of the 33 available web servers. Each HTTP request gathered contains: timestamp of the request, client ID who made the request, ID of the requested item, bytes of the response, method, status code, format of file requested, and server ID which handled the request.

For our simulation experiments we used the June 9th logs of the 1998 World Cup data set, which contained approximately 20 million web requests. On this day only 26 of the 33 web servers were active, thus we used 26 monitoring nodes. To goal of our monitoring process was to track frequently requested items by users. Similarly to the Lawrence Berkeley data set, we did not need all the data fields provided. Therefore, we created a new text file containing only the timestamp, requested item ID, and server ID which handled the request of each record.

4.2 Input Parameter Summary

The Top-K Monitoring method required a series of user defined parameters in order to monitor the top-*k* elements effectively. Since our monitoring approach is an extension, we too must define a series of parameters. Each of these parameters may affect the performance of our method in a variety of ways, therefore, must be examined in our experimental evaluation. These parameters are briefly reviewed in this section.

4.2.1 Support Parameter

In our definition of the frequent items problem given in Section 1.2.2 the user must define a support parameter s. This support parameter affects the threshold used to define when an item is to be characterized as frequent. Theoretically, there can be at most $\frac{1}{s}$ frequent items (Cormode and others, 2005). From this property it is clear that when the support parameter is lowered more items can potentially be classified as frequent. When the support parameter is raised, however, fewer items can potentially be classified as frequent.

4.2.2 Error Tolerance Parameter

In Section 3.3 we introduced methods to reduce memory requirements by integrating an ε-approximate summary data structure onto each monitoring node. The amount of error was found to be bounded by a user defined error tolerance parameter ε. The higher the value of ε, the less trustworthy the frequency counts are for a given item. However, the benefit gained from this is in reduction of memory required. Although this parameter can be set to any value less than s, it is generally given the value of 10% the support parameter (Manku and Motwani, 2002).

4.2.3 Reallocation Parameters

Recall from Section 3.2.2, that the final step of reallocation is to assign a portion of Δ_j to each new adjustment factor corresponding to counter $c_{j,i}$ on monitoring node N_i. The amount assign to each is depended upon user defined allocation parameters $0 \leq F_i < 1$. The values of these allocation parameters can be assigned in a variety of different ways, as long as, $\Sigma_{0 \leq i \leq m} F_i = 1$. Babcock and Olston (2003) addressed this issue when evaluating Top-K Monitoring and provided the following heuristics.

To simplify the process of setting the allocation parameters the authors first assigned F_0, which is defined as the coordinator allocation parameter. Once this is done the remaining parameters are allocated a portion of the remaining $1 - F_0$. The fashion in which the remaining portion was allocated is based upon one of two proposed methods:

1. *Even Allocation*: The remaining $1 - F_0$ is even distributed to each monitoring node in η.

2. *Proportional Allocation*: Each node is allocated a portion of $1 - F_0$ proportional to the amount of traffic observed locally. That is, a larger portion is allocated to nodes which experience larger volumes of traffic.

4.3 Performance Criteria

To evaluate our frequent item monitoring approach a series of performance criteria was determined. We believe each of the criteria defined reflect our goals in this thesis. The three criteria used include: communication cost, approximation accuracy, and memory requirements. A detail description of each of these criteria and how they were measured is given in the following subsections.

4.3.1 Communication Cost

We defined communication cost as the ratio of the number of bits communicated between any two nodes over the number of bits transmitted using a centralized monitoring approach. In a centralized approach we assume only the item identifier for an update tuple is communicated when no sliding window is involved. The item identifiers in our data sets can each be represented as a 32-bit integer. If a time-based sliding window is utilized, we must also include a 32-bit timestamp.

Communication is only conducted with our approach during resolution. Thus the total number of bits transmitted can be calculated as the total number of bits sent during each individual resolution phase. The number of bits transmitted during any single resolution phase (BPR) is given in Equation 1.

$$EPR = |\Im|\cdot|\eta|\cdot(32+32+64)+|\Im|\cdot|\eta|\cdot(32+64)+|\eta|\cdot 64 \qquad (1)$$

In Equation 1, $|\Im|$ is defined as the total number of broken constraints which resulted in the resolution. We assume that the local thresholds and the adjustment factors account for 64-bits each.

4.3.2 Approximation Accuracy

In Section 3.3 we introduced methods to reduce memory requirements by weakening the problem definition. More precisely, we allowed a bounded amount of error on each frequency count and provided the set of approximate frequent items. Although the amount of error is guaranteed to be no more than a user defined error tolerance, the accuracy of the approximate frequent item set has no guarantees.

To measure the actual quality of the approximations, we adopted two commonly used criteria. These include precision and recall. Precision is defined as the percentage of correct items contained in the entire output. Similarly, recall is defined as the percentage of correct items contained in the output to the number of total possible correct items (Cormode and Muthukrishnan, 2003).

It is sometimes helpful to combine these two measurements into a single value. C.J. van Rijsbergen (1979) gives us a measurement called an F-measure, which weighs precision and recall equally. This equation gives the overall quality of the output and can be expressed as follows:

$$F - Measure = \frac{2 \cdot P \cdot R}{(P + R)} \qquad (2)$$

4.3.3 Memory Requirements

In our monitoring approach a number of frequency counts are collected for each observed item. The bulk of memory required by our approach is dominated by the number of these counters. Although we bounded worst case memory requirements in Section 3.3.4, we believe that the actual number of counters used may be significantly less. Therefore, we measured memory requirements as the maximum number of counters stored on any monitoring node at any given time.

4.4 Experimental Results

For each of the performance criteria defined in the previous section, we ran a series of experimental simulation runs. Varies input parameters were used in each experiment to see their affects on these criteria (communication cost, approximation accuracy, and memory requirement). The set-up and individual descriptions of each experiment is given in the following subsections. Additionally, some heuristics are introduced to give a better understanding on how the input parameters should be assigned.

4.4.1 Communication Cost

Our first set of experiments examined the communication cost incurred using the two data set described in Section 4.1. In each simulation run we varied the support parameter s, held the error tolerance fixed at $\varepsilon = 0$, and varied the coordinator allocation parameter F_0. For all simulation runs we used even allocation to assign the remaining allocation parameters. The reason for this is that preliminary results showed no significant differences in the communication cost using either discussed approaches. These results were in agreement with the analysis conducted by Babcock and Olston (2003) in their experimentations with Top-K Monitoring. Finally, we initiated our monitoring approach on the first update tuple. Therefore, no special initialization phase was used. In Figure 4.1 and Figure 4.2 we see the results of these initial experiments.

The results show that the effects of the coordinator allocation parameter F_0 on communication cost differs between the two data sets. In the Lawrence Berkeley TCP data set we see that when the allocation parameter is increased, communication cost also increases. The opposite pattern occurs with the 1998 World Cup data set.

As was seen in the analysis of Top-K Monitoring by Olston and Babcock (2003), when $F_0 > 0$ reallocation can prevent reaching the expensive (re)synchronization phase. However, this comes at the cost of having more fragile constraints which may break more frequently. This same scenario also occurs with our monitoring approach and can explain the differences we see.

Since only four monitors were used with the Lawrence Berkeley TCP data set, the (re)synchronization phase required little communication and the weaker constraints could not offset this cost. In contrast, the 1998 World Cup data set consisted of 26 monitoring nodes. Thus the (re)synchronization phase was much more expensive, requiring many nodes to be contacted. From these results we recommend that F_0 be assigned a small value (< 0.3) when there are few nodes and a large value when there are many.

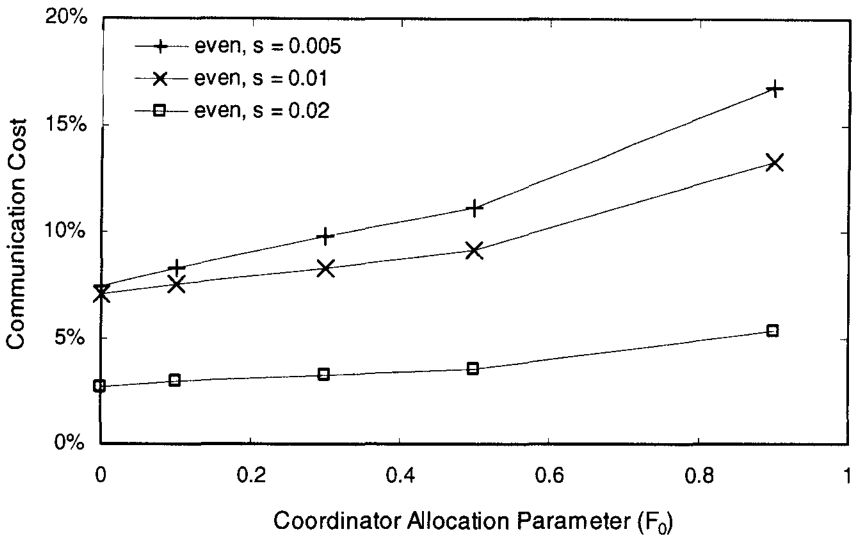


FIGURE 4.1 – Communication cost for Berkeley TCP data set.

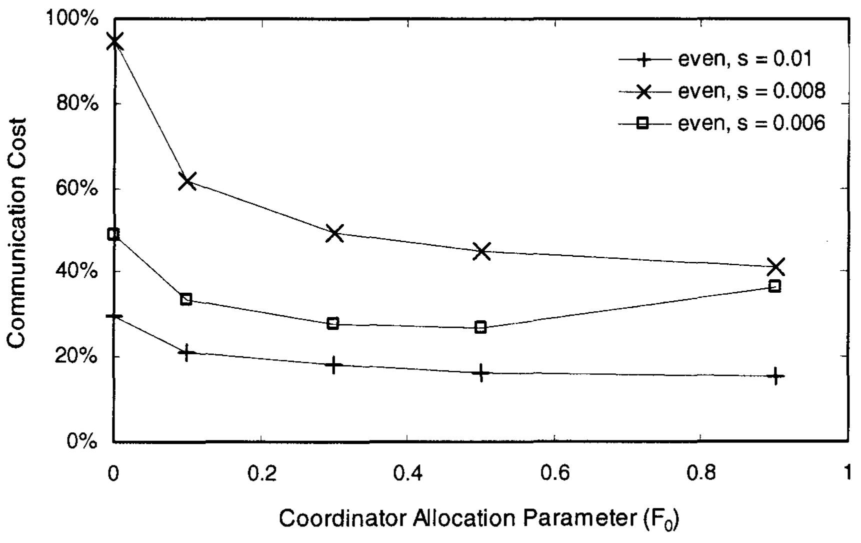


FIGURE 4.2 – Communication cost for ’98 World Cup data set.

Both results show that by reducing the support parameter communication cost increases. This is not surprising since in theory, as the support is decreased the expected size of the frequent item set increases. An anomaly did occur, however, in the 1998 World Cup data set when $s = 0.008$. In this scenario it is assumed that the frequent item set becomes more volatile. This demonstrates the need for the data to maintain a degree of stability in order for our purposed method to significantly reduce communication cost.

Our second experiment focused on how communication cost accumulated over time to reach its final value. The 1998 World Cup data set was used for this experiment but execution was terminated after 500,000 update tuples. The coordinator reallocation parameter was set fixed at $F_0 = 0$, while we varied the support parameter. To determine how communication accumulates we held constant the number of update tuples to the total number in the data set (approximately 20 million) in our communication cost formula. The results of this experiment are shown in Figure 4.3.

We see from our results a sudden spike in communication cost occurring during the initial 100,000 update tuples. Afterwards, communication only steadily rises until reaching its final value. This sudden spike is most extreme when $s = 0.008$. In this case over 12 million bytes were transmitted in the first 100,000 update tuples alone. As a result, we believe if an initialization phase was used to account for these first 100,000 tuples, overall communication cost can be reduced significantly.

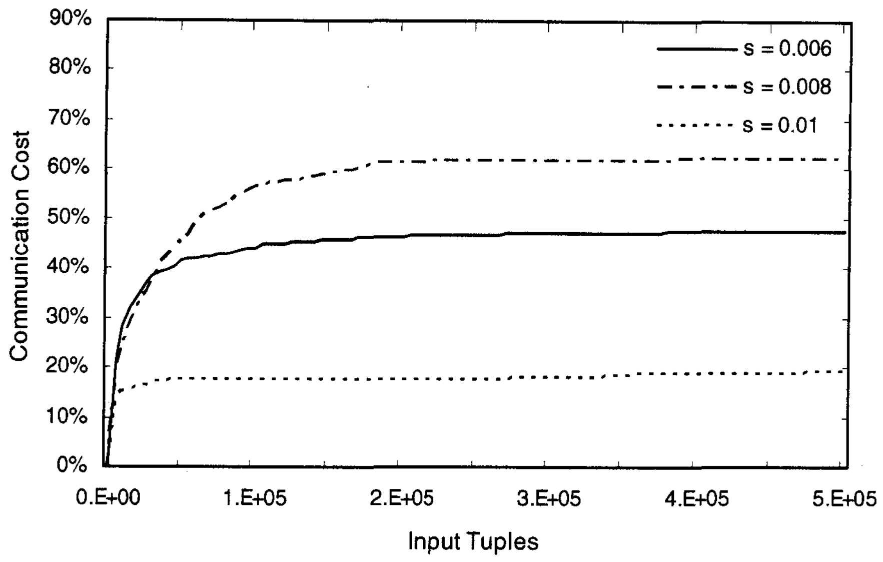


FIGURE 4.3 – Communication cost over time for '98 World Cup data set.

Finally, our third experiment focused on communication cost required when the error tolerance $\varepsilon = 0.1 \cdot s$. We saw, in this scenario, that communication cost was between -1.5% and 0.5% from the results given when the error tolerance $\varepsilon = 0$. That is, in some cases communication was insignificantly raised and others were insignificantly lowered. This signifies that our adjustment factor maintenance policies are both lightweight and communication efficient.

4.4.2 Approximation Accuracy

Accuracy of our monitoring method is directly depended upon the accuracy of the summary data structured integrated into our monitoring nodes. As a result, to study the approximation accuracy is to study the actual summary data structure utilized. Recall that we adopted the MG algorithm and modified the threshold in hopes of increasing the accuracy of the counting technique. Therefore, our experiments focus on examining the accuracy of both the original MG algorithm and our modified version.

To examine the accuracy of both versions of the MG algorithm, we conducted a series of runs using the Berkeley TCP data set. These runs were done on a centralized node environment; however, similar results are expected when placed in a distributed setting. During each run we measured the average precision of both the unmodified MG algorithm and our modified threshold version. The results of these runs are given in Figure 4.4 below.

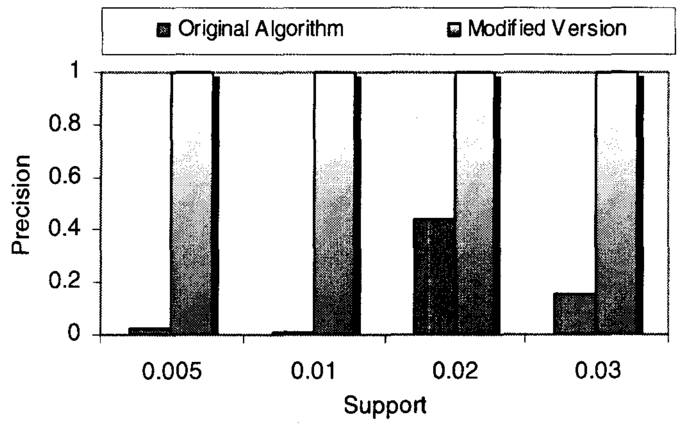


FIGURE 4.4 – MG algorithm average precision.

We see from the results that significant improvement was made on the precision of the MG algorithm. This is most extreme when $s = 0.01$, which showed a 99% increase in precision. The least improvement was found to be when $s = 0.02$, which still showed an increase in over 50%.

These results are not surprising, however, when examining the threshold used by the traditional MG algorithm. Recall in the traditional MG algorithm, we subtracted $\varepsilon \cdot N$ from the threshold to account for the maximum possible amount of error introduced to each frequency count. To demonstrate this over-estimation in the error introduced we ran a series of runs with varying supports and reported the true number of decrements, or the true error introduced on the frequency counts, at the end of each run. The results in Table II clearly show this over-estimation. Thus counting the true number of decrements (or true error introduced) can greatly improve the accuracy of the MG algorithm.

TABLE II

DECREMENTS OBSERVED IN THE MG ALGORITHM

Support	Max Decrements	True Decrements
0.005	8949	0
0.01	17899	31
0.02	35799	313
0.03	53699	951

4.4.3 Memory Requirements

To provide the exact frequent item set, a counter for each observed item must be kept in memory. The amount of memory in this case is bounded by |U| which may be very large. In the Lawrence Berkeley TCP data set there was 1,622 unique user IDs. The much larger 1998 World Cup data set contained 9,198 unique item IDs. Thus we expect each monitor to require equal amount of counters.

When tracking the approximate frequent item set, we saw in Section 3.3.4 that each monitoring node required between $O(\frac{1}{\varepsilon})$ and $O(\frac{m}{\varepsilon})$ counters. To test the actual number of counters used we setup a series of runs with varying supports using both the Berkeley TCP data set and the 1998 World Cup data set. The results of these runs are given in Table III.

TABLE III

MEMORY REQUIREMENTS FOR APPROXIMATE TRACKING

	Support	Max Counts (m/ε)	Max Counts ($1/\varepsilon$)	Max Counts (Actual)
World Cup Data Set	0.01	26,000	1,000	1,281
	0.008	32,500	1,250	1,521
	0.006	43,334	1,667	1,895
Berkeley Data Set	0.02	2,000	500	502
	0.01	4,000	1,000	1,001
	0.005	8,000	2,000	1,489

Given in Table III are the maximum counts used by any monitor, as well as, the worst cases bound for our method $O(\frac{m}{\varepsilon})$. Also included is the worst case memory requirements for any centralized monitoring approached $O(\frac{1}{\varepsilon})$ for comparison. Ideally, our method would use approximately the same amount of memory as any centralized approached in practice. Our results verify that this is true, since the number of counters used is much closer to $\frac{1}{\varepsilon}$ than it is to $\frac{m}{\varepsilon}$. In one case the number of counters used was found to be less than the worst case bound for any centralized approach.

4.5 Comparison

To finalize our evaluation, we compared our frequent item monitoring approach with Top-K Monitoring. We measured the communication cost and the output quality of both methods using exact counting. Since our monitoring method is designed explicitly to solve the frequent items problem and we are doing exact counting, the accuracy is perfect. That is, there are no false positives and no false negatives, yielding an F-measure of 100%.

Top-K Monitoring was not originally designed to monitor frequent items, but rather to provide the top-k ranked items. Recall in Section 1.2.4, that if $k = \frac{1}{s}$ we are guaranteed to report all frequent items. Thus, we can use any top-k monitoring method to also monitor frequent items. Since we use very small support values, however, assigning k in this fashion would yield a very large list outputted. This list is also likely to contain a large number of false positives (or a low precision value). Therefore, before comparison we first investigated various setting of k with the goal of improving overall output quality.

Our experiments used Lawrence Berkeley TCP data set. We measure both recall and precision every 100,000 update tuples and formed an average of these measurements. These two averages were then combined to form an F-measure. We plotted these measures on a curve given in Figure 4.5. We see from these results that the overall quality of the output can be greatly improved using a $k < \frac{1}{s}$.

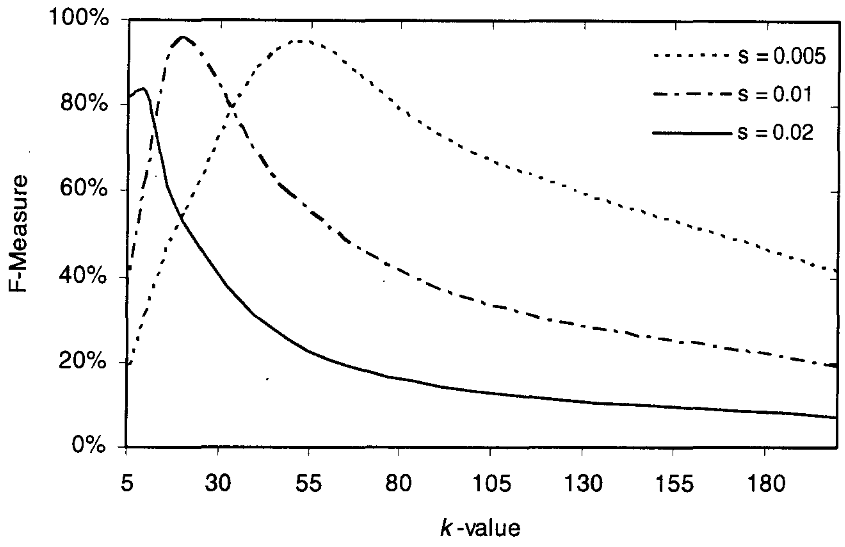


FIGURE 4.5 – Output quality of Top-K Monitoring on Berkeley TCP data set.

Using our results above as a guide for selecting an appropriate value for *k*, we then began comparing Top-K Monitoring with our modified version. In these experiments we again used the Lawrence Berkeley TCP data set. We varied the coordinator allocation parameter and the support value in each simulation run and measured the final communication cost. The allocation parameter which yielded the optimal communication cost was selected in both methods. Additionally, we selected the *k* value which yielded the optimal F-measure for each support. The results of our comparisons are given in Table IV.

We see from the results that in all scenarios communication cost was lower using our modified explicit version. The difference between the two methods was greatest when the support is low. This is not surprising since when the support is low, the required k needed to report all frequent items must be high. Since the entire top-k set is transmitted when any constraint is broken, a large k will also yield very long messages. This is likely the reason for the large gap in communication cost. Additionally, we see from the results that the k value chosen approximates the average size of F outputted for each support. Thus the differences in communication cost are caused more by the differences in the actual algorithms rather than by the output sizes.

TABLE IV

COMPARISON RESULTS WITH TOP-K MONITORING

Method	Support	Avg. Output Size	Communication	F-Measure
Top-K Monitoring	0.005	50.00	143.34%	95.37%
	0.01	20.00	46.66%	96.07%
	0.02	10.00	12.20%	83.62%
Modified Version	0.005	52.25	7.43%	100.0%
	0.01	21.45	7.03%	100.0%
	0.02	7.48	2.70%	100.0%

For our final comparison we observed the scalability of each method. To do this we varied the number of monitoring nodes using the Lawrence Berkeley TCP data set. In each simulation we held the support fixed at 0.01 and used k = 20. The results of this experiment shows that communication cost for both methods grow approximately linear to the number of monitoring nodes. However, with our modifications the rate of growth is much less.

CHAPTER 5

APPLICATION: DETECTING DDoS ATTACKS

5.1 Introduction

Denial of service (DoS) attacks is a malicious attempt by a single person or a group of people to prevent legitimate users from accessing a provided service (cert.org, 1997) Although DoS attack patterns are highly diversified, coming in many different forms, the most generally observed pattern involves the transmission of numerous packets toward a single destination (Houle and Weaver, 2001). The goal is to overload the available bandwidth, or other resources of the intended victim, with a surge of network traffic. These types of attacks are commonly called bandwidth attacks and consists of TCP flooding, ping flooding, and UDP flooding.

Previously the most common DoS attacks were conducted by a single source directed at a single victim (Houle and Weaver, 2001). Since the year 1999, however, many more sophisticated attacking tools have been created which utilize multiple attacking sources. These newly introduced attacking patterns have created an additional class of its own, called distributed denial of service attacks (DDoS). The goal of these attacks is the same as its predecessors, but overwhelms the resources of the intended victim by brute force numbers.

Generally a DDoS attack consists of two stages, each depicted by Figure 5.1. The first stage consisted of the identification and infiltration of numerous host computers. These subverted machines are usually called "zombies", although also referred to as agents. Once a desired number of agents have been gathered, the final step is for the actual attack to begin. The attack traffic is generated by each agent, and is propagated toward the victim. To hide the identity of each agent, in hopes of preventing easy detection, some of the agents may use spoofed addresses. Luckily for many defense systems, the scenario of spoof addressing is found to be rare in actual practice (Mao and others, 2006).

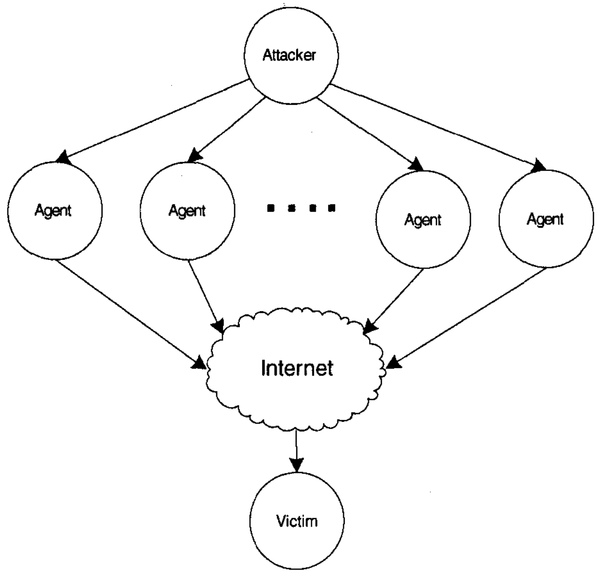


FIGURE 5.1 – DDoS attacking pattern.

The tools used to launch DDoS attacks are relatively simple to use and readily available (Zhang and Parashar, 2005). This has resulted in a surge of attacks recently, some even amongst top web companies. For example in the year 2000, websites such as

Yahoo, Amazon.com, eBay, and others all experienced regional outages caused by DDoS attacks (cnn.com, 2000). In 2003 eBay received tens of thousands of dollars in damages from a 20,000 agent coordinated attack on their website (spamdailynews.com, 2005).

5.2 Detecting DDoS Attacks

Motivated by the impact of DDoS attacks on the Internet community, we will examine how our frequent item monitoring method can be used as a detection system. As stated, DDoS attacks are generally characterized by a surge of network traffic toward an intended victim host. Past detection solutions have used this simple, but obvious characteristic, as a detection criterion (Akella and others, 2003; Manjhi and others, 2005; Sekar and others, 2006). Generally, these systems track destination addresses which have received a disproportional amount of traffic in the observed network.

Similar to past solutions, we will continuously monitor destination addresses receiving a large number of packets over a given time. That is, we will report frequently used destination addresses. Since DDoS attacks have only a limited duration, we must maintain time-sensitive frequency counts. To do this we buffered all update tuples occurring over the last five minutes, which is the commonly observed attack duration in the Internet (Moore, Voeker, and Savage, 2001.).

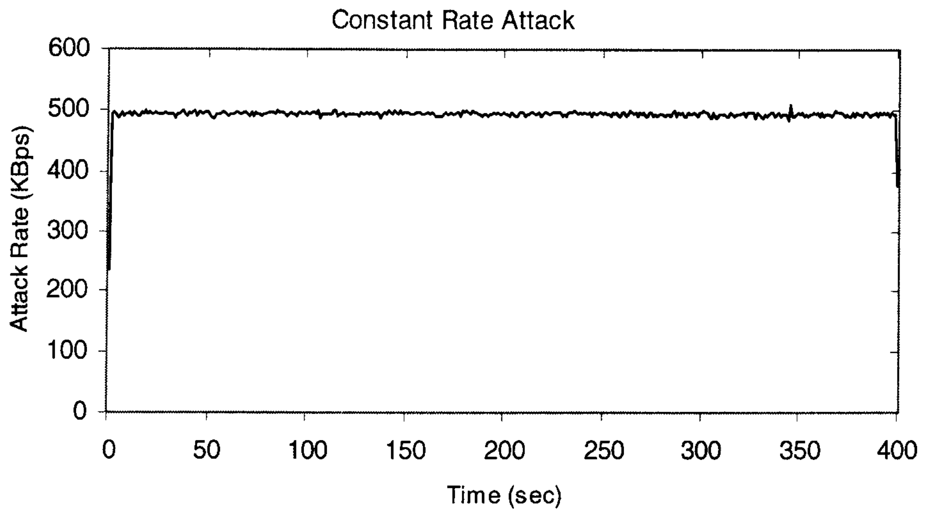
Since any detection system has an inherited degree of inaccuracy, we used exact counting. By examining the detection capabilities in this fashion, we will know immediately if our approach is applicable as a detection tool. If the detection capabilities are then found to be satisfactory and memory requirements become a concern, the methods provided in Section 3.4 can be utilized. However, approximation frequent counts may degrade the detection accuracy of our method.

5.3 Data Sets

To evaluate the applicability of our monitoring method as a possible DDoS detection tool, we adopted the publicly accessible UCLA CSD network traces (lever.cs.ucla.edu). Available network traces include normal TCP and UDP traffic traces collected at a border router. Additionally, a collection of attack traces were generated from a testing machine using the tfn attack tool. Information gathered on each packet include: timestamp, source address, destination address, source port, destination port, and length packet in bytes. All addresses were renumbered using a collection of scripts to preserve the privacy of the network users.

To create a more complete data set, we a combined a series of normal traffic traces with four different types of DDoS attack traces. The four different attack patterns include (Mirkovic, Prier, and Reiher, 2002):

1. *Constant Rate Attack*: Represents the majority of attack patterns, which deploys a continuous maximum rate of packets.
2. *Pulsing Attack*: Every 100 seconds the attack rate will oscillate from the maximum rate to zero.
3. *Increasing Rate Attack*: The attack rate increases over a 300 second time-span until the maximum rate is achieved.



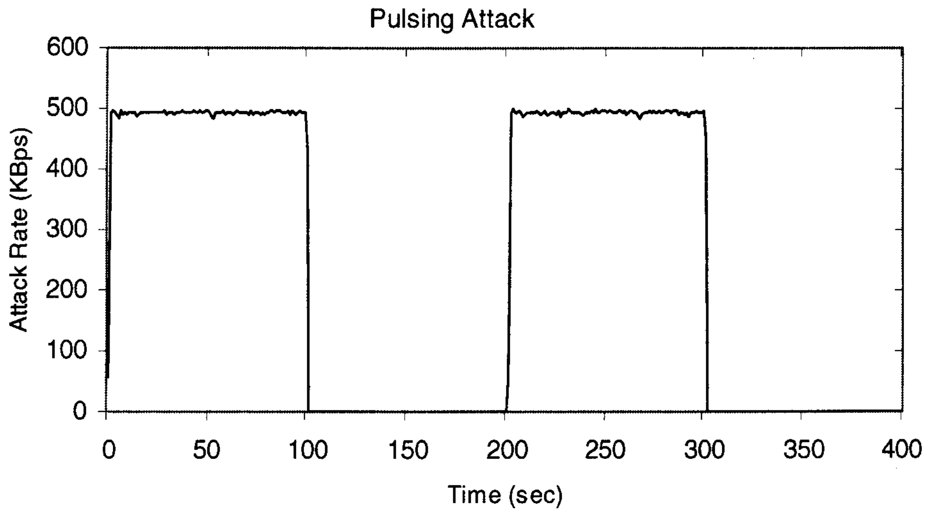
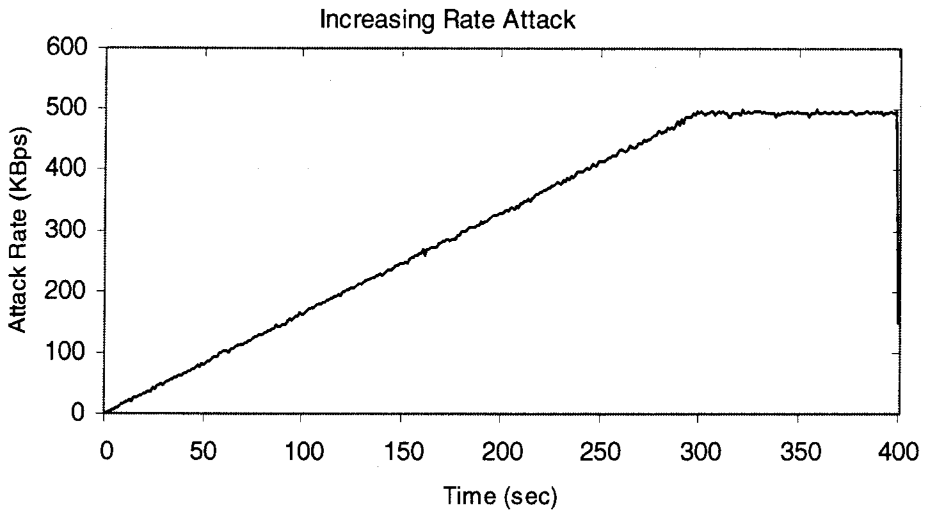


FIGURE 5.2 – Constant and pulsing rate attack pattern.



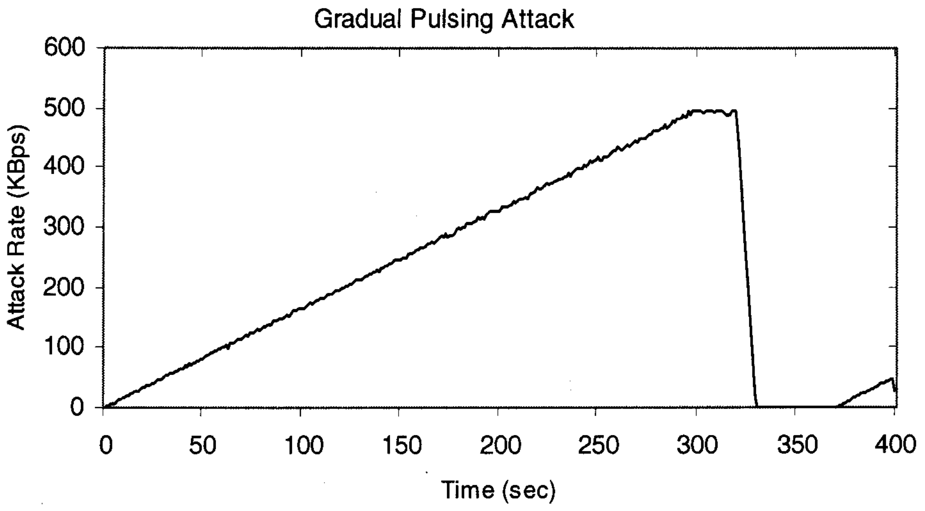


FIGURE 5.3 – Increasing and gradual pulsing attack pattern.

4. *Gradual Pulse Attack*: The attack rate increases over a 300 second time-span. When the maximum rate is achieved, it is maintained for 20 seconds then gradually decreased to zero over 10 seconds. The pattern then repeats after a 40 second inactive period.

Each attack pattern used had a maximum rate of 500KBps. The attack patterns are represented in Figure 5.2 and Figure 5.3. Similar to the Lawrence Berkeley data set we simulated a distributed environment by evenly assigning packets to four monitoring nodes.

5.4 Performance Criteria

We measured the performance of our system using four common intrusion detection system (IDS) evaluation criteria. Statistics reflecting these criteria were measured during each of our simulation runs. The criteria we used and their definitions are given as follows:

1. *Detection Delay*: The amount of time between when the attack begins and when the address of the victim becomes frequent. When the address of the victim becomes frequent we say this signifies an attack. This delay does not include communication delay or processing delay.
2. *Detection Rate*: The percentage of the number of attacks detected to the total number of attacks in the data set.
3. *False Positive Rate*: The percentage of the number of packets received at a destination wrongly classified as a victim of an attack to the total number of packets in the data set.

4. *Communication Cost*: The number of bytes communicated by all the monitoring nodes over the number of bytes needed to forward all item occurrences and their timestamps to a centralized location for processing

5.5 Experimental Results

In each simulation run we varied the support parameter s, held the error tolerance fixed at $\varepsilon = 0$, and varied the coordinator allocation parameter F_0. The coordinator allocation parameter which yielded optimal communication cost was reported in all our results. Statistics needed for each criterion given, was gathered during each run. The results of our experiments are summarized in Table V below.

TABLE V

DDOS DETECTION EVALUATION RESULTS

Support	Detection Rate	False Positive Rate	Average Delay (seconds)	Communication Cost
0.2	100.0%	12.99%	26.51	7.11%
0.3	100.0%	5.10%	35.98	0.99%
0.4	100.0%	2.84%	46.42	0.41%
0.5	100.0%	2.24%	58.84	0.24%

We see that with an appropriately set support value all attacks can be detected in less than a minute. Additional false positives may also occur, however, these alarms may still be of interest since they represent destinations receiving significant amount of traffic over a short period of time. Finally, we see communication cost is significantly lower than forwarding all traffic events to a centralized location for analysis.

This preliminary evaluation shows promising results when compared with three other known DDoS detection methods (Chen and Hwang, 2006; Peng, Leckie, and Ramamohanarao, 2004; Zhang and Parashar, 2005). Table VI gives a comparison table of all methods examined. We see from the table that each detection method yielded a high detection rate of 75-100%. However, similar to our results, methods (Chen and Hwang, 2006) and (Zhang and Parashar, 2005) also yielded some false alarms. Our false positive rate is most comparable with (Zhang and Parashar, 2005) which showed a rate between 7.67% and 12.12%. The work in (Chen and Hwang, 2006) examined different threshold values and traded off detection accuracy to reduce false positives to less than 1%. Only (Peng, Leckie, and Ramamohanarao, 2004) measured the detection delay of their method, which required between 69.7 and 10 seconds depending on the aggressiveness of the attack. Although our detection delays are comparable, we do not include processing and communication delay.

TABLE VI

COMPARISON OF MUTLIPLE DDoS DETECTION SYSTEMS

Method	Detection Rate	False Positive Rate	Average Delay (sec)
(Chen *et al.*, 2006)	75-100%	0-70%	No data
(Peng *et al.*, 2004)	90-100%	No data	10-69.7
(Zhang *et al.*, 2005)	No data	7.67-12.12%	No data
Proposed System	100%	2.24-12.99%	26.51-58.84

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis we examined the problem of monitoring frequent items over distributed data streams. The original goal was to create a system that used limited memory, computation time, and communication. We examined a variety of prior solutions in the problem domain and in closely related domains. One solution which gained particular attention was the communication efficient Top-K Monitoring method proposed by Babcock and Olston (2003). It was determined that their method could be modified to explicitly monitor frequent items.

Although the original Top-K Monitoring approach allowed for approximate answers, they did not utilize summary data structures. As a result, memory requirements were still dominated by the number of unique items observed. To fuse two prior methodologies together, we integrated a modified version of the MG algorithm into our monitoring approach. This allowed us to reach our initial goal of requiring limited memory space.

To evaluate all of our modifications, we used two widely used and publicly available data sets. The results of our experiments demonstrated that our approach used less communication and scaled better then the original Top-K Monitoring method.

Additionally, approximate answers were provided with near exact results. This was accomplished while still requiring a limited amount of memory space.

Finally, we examined the important problem of detecting DDoS attacks in a networked environment. Since DDoS are characterized by a flood of network traffic, this problem fits appropriate with our monitoring task. After evaluation, our findings demonstrated that our method can indeed be used as a detection tool. When compared with other known detection systems, we saw that our approach yielded favorable results.

6.2 Future Work

Our evaluation results showed two important issues with our monitoring method. First although our approach yielded better scalability than Top-K Monitoring, it still scaled linearly to the number of monitoring nodes. More analysis is needed in this direction to determine the performance under a large number of monitoring nodes (>100). Second, we saw that a significant amount of communication cost can be avoided if an initialization phase is utilized. Further analysis is needed to study different initialization methods and heuristics on how long they should operate before our monitoring method can begin.

In the selected application domain we saw that our method yielded favorable results. Our preliminary analysis, however, should still be expanded in more detail. This would include implementing our monitoring approach on a specialized networking simulator. More detailed analysis can be conducted in this fashion. This is important since our analysis ignored the contributions of propagation and transmission times in the detection delay. Additionally, deployment issues were not addressed in our work. This could include the cost of implementing in a real world ISP network and how likely it would gain acceptance in the IDS community.

Finally, in many application domains there is a need for time sensitive data. That is, to weigh more recent occurrences more than older ones. Although this topic was covered in this thesis, we did not provide a comprehensive evaluation. More analysis in this direction is therefore needed.

REFERENCES

Aggarwal, C., Parthasarathy, S., Ghoting, A., and Otey, M. 2006. *Data Streams: Models and Algorithms.* New York: Springer-Verlag.

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

Arasu, A. and Manku, G. 2004. Approximate counts and quantiles over sliding windows. *Proceedings of the 23rd Symposium on Principles of Database Systems (Paris, France),* 286-296.

Babcock, B., Babu, S., Datar, M., and Motwani, R., and Widom, J. 2002. Models and issues in data stream systems. *Proceedings of the 21st Symposium on Principles of Database Systems (Madison, Wisconsin),* 1-16.

Babcock, B. and Olston, C. 2003. Distributed top-k monitoring. *Proceedings of the 2003 AMC SIGMOD International Conference on Management of Data (San Diego, California),* 28-39.

Chen, Y. and Hwang, K. 2006. Collaborative change detection of DDoS attacks on community and ISP networks. *Proceedings of the International Symposium on Collaborative Technologies and Systems (Las Vegas, Nevada),* 401-410.

Cohen, E. and Strauss, M. 2003. Maintaining time-decaying stream aggregates. *Proceedings of the 22nd Symposium on Principles of Database Systems (San Diego, California),* 223-233.

Cormode, G. and Garofalakis, M. 2005. Sketching streams through the net: distributed approximate query tracking. *Proceedings of the 31st International Conference on Very*

Large Data Bases (Trondheim, Norway), 13-24.

Cormode, G. and Garofalakis, M. 2005. Efficient strategies for continuous distributed tracking tasks. *IEEE Data Engineering Bulletin*, 25:33-39.

Cormode, G., Garofalakis, M., Muthukrishnan, S., and Rastogi, R. 2005. Holistic aggregates in a networked world: distributed tracking of approximate quantiles. *Proceedings of the International Conference on Management of Data (Baltimore, Maryland)*, 25-36.

Cormode, G. and Muthukrishnan, S. 2003. What's hot what's not: tracking most frequent items dynamically. *Proceedings of the 22nd Symposium on Principles of Database Systems (San Diego, California)*, 296-306.

"Cyber-attacks batter Web heavyweights." February 2000, available from http://archives.cnn.com/2000/TECH/computing/02/09/cyber.attacks.01/index.html; accessed 8 March 2007.

Demaine, E., Lopez-Ortiz, A., and Munro, J. 2002. Frequency estimation of internet packet streams with limited space. *Proceedings of the 10th Annual European Symposium on Algorithms*, 348-360.

"Denial of Service Attacks." October 1997, available from http://www.cert.org/tech_tips/denial_of_service.html; accessed 8 March 2007.

Golab, L., DeHaan, D., Demaine, E., Lopez-Ortiz, A., and Munro, J. 2003. Identifying frequent items in sliding windows over on-line packet streams. *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (Miami Beach, Florida)*, 173-178.

Golab, L. and Ozsu, M. 2005. Update-Pattern-Aware modeling and processing of continuous queries. *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (Baltimore, Maryland)*, 658-669.

Hinden, R. and Deering S. "IPv6 Addressing Architecture." February 2006, available from http://tools.ietf.org/rfc/rfc4291.txt; accessed 5 March 2007.

Houle. K. and Weaver, G. "Trends in Denial of Service Attack Technology". October 2001, available from http://www.cert.org/archive/pdf/DoS_trends.pdf; accessed 8 March 2007.

Karp, R., Shenker, S., and Papadimitriou, C. 2003. A simple algorithm for finding frequent items in streams and bags. *ACM Transactions on Database Systems*, 28:51-55.

Keralapura, R., Cormode, G., and Ramamirtham, J. 2006. Communication-efficient distributed monitoring of threshold counts. *Proceedings of the International Conference of Management of Data (Chicago, Illinois)*, 289-300.

Kim, H. and Karp, B. 2004. Autograph: Toward automated distributed worm signature detection. *Proceedings of the 13th USENIX Security Symposium (San Diego, California)*, 271-286.

Lee, L.K. and Ting H.F. 2006. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. *Proceedings of the 25th Symposium on Principles of Database Systems (Chicago, Illinois)*, 290-297.

Li, H., Lee, S.Y., and Shan, M.K. 2005. Online mining (recently) maximal frequent itemsets over data streams. *Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, 11-18.

Liu, H., Lu, Y., Han, J., and He, J. 2006. Error-adaptive and time-aware maintenance of frequency counts over data streams. *Proceedings of the 7th International Conference on Web-Age Information Management (Hong Kong, China)*, 484-495.

Madden, S., Franklin, M., Hellerstein, J., and Hong, W. 2003. The design of an acquisitional query processor for sensor networks. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (San Diego, California)*, 491-502.

Manjhi A., Shkapenyuk, V. Dhamdhere, K., and Olston, C. 2005. Finding (recently) frequent items in distributed data streams. *Proceedings of the 21st International*

Conference on Data Engineering, 767-778.

Manku, G. and Motwani, R. 2002. Approximate frequency counts over data streams. *Proceedings of the 28th International Conference on Very Large Data Bases (Hong Kong, China)*, 364-357.

Mao, Z., Sekar, V., Spatscheck, O., van der Merwe, J., and Vasudevan, R. 2006. Analyzing large DDoS attacks using multiple data sources. *Proceedings of the 2006 SIGCOMM Workshop on Large-Scale Attack Defense (Pisa, Italy)*, 161-168.

Metwally, A., Agrawal, D., and Abbadi, A. 2005. Computation of frequent and top-k elements in data streams. *Proceedings of the 10th International Conference on Database Theory (Edinburgh, Scotland)*, 398-412.

Mirkovic, J., Prier, G., and Reiher, P. 2002. Attacking DDoS at the source. *Proceedings of the 10th IEEE International Conference on Network Protocols (Paris, France)*, 312-321.

Misra, J. and Gries, D. 1982. Finding repeated elements. *Science of Computer Programming*, 2:143 – 152.

Moore, D., Voeker, G., and Savage, S. 2001. Inferring Internet denial-of-service activity. *Proceedings of the 10th USENIX Security Symposium (Washington, D.C.)*, 9-22.

Peng, T., Leckie, C., and Ramamohanarao, K. 2004. Proactively detecting distributed denial of service attacks using source IP address monitoring. *Proceedings of the 3rd International Networking Conference (Athens, Greece)*, 771-752.

"Sanitized UCLA CSD traffic traces." available from http://lever.cs.ucla.edu/ddos/traces/; accessed 8 March 2007.

Sekar, V., Duffield, N., Spatscheck, O., van der Merwe, J., and Zhang, H. 2006. LADS: Large-scale automated DDoS detection system. *Proceedings of USENIX Annual Technical Conference (Boston, Massachusetts)*, 171-184.

Stanojevic, R. "Scalable heavy-hitter indentification." available from http://www.hamilton.ie/person/rade/ScalableHH.pdf; accessed 4 March 2007.

Sun, J., Papadimitriou, S., and Faloutsos, C. 2006. Distributed Pattern Discovery in Multiple Streams. Technical Report CMU-CS-06-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Yu, H., Li, H.G., Wu, P., Agrawal, D., and Abbadi, A. 2005. Efficient processing of distributed top-k queries. *Proceedings of the 16th International Conference on Database and Expert System Applications (Copenhagen, Denmark)*, 65-74.

Zhang, G. and Parashar, M. 2005. Cooperative defense against DDoS attacks. *Proceedings of the 3rd International Workshop on Security in Information Systems (Miami, Florida)*, 113-122.

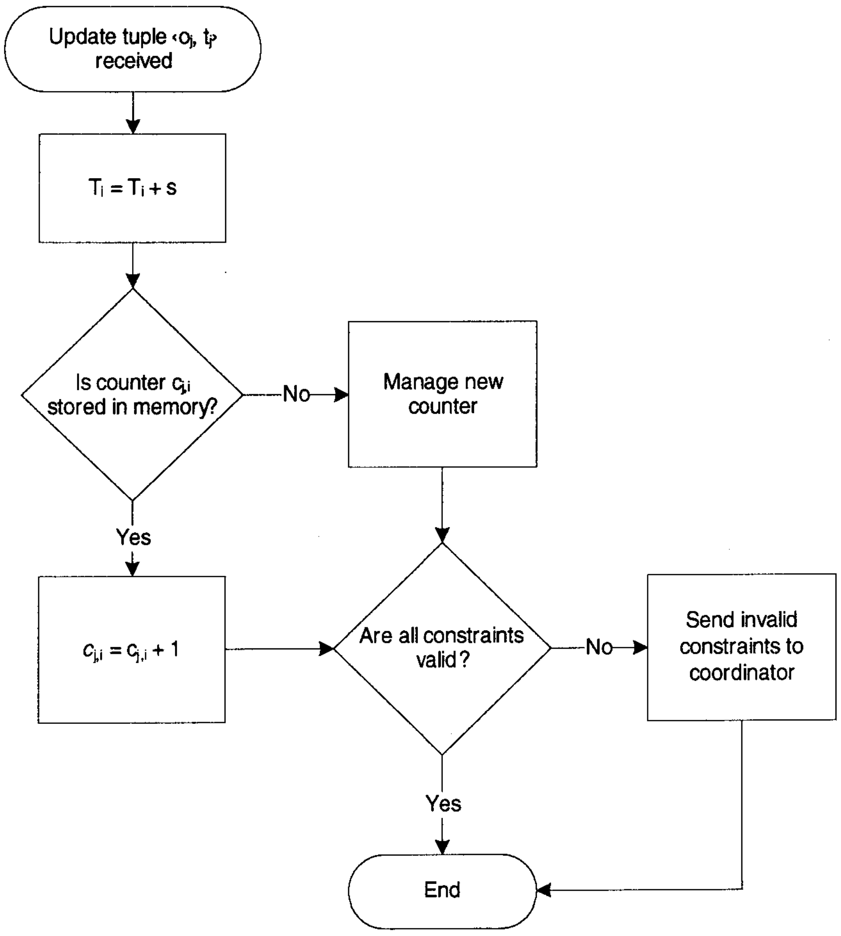
Zhu, Y. and Shasha, D. 2002. StatStream: Statistical monitoring of thousands of data streams in real time. *Proceedings of the 28th International Conference on Very Large Databases (Hong Kong, China)*, 358-369.

"Zombie master pleads guilty of eBay Internet attack." December 2005, available from http://www.spamdailynews.com/publish/Zombie_master_pleads_guilty_to_ eBay _Internet_attack.asp; accessed 8 March 2007.

APPENDIX I

MONITORING NODE UPDATE PROCESS

The flow chart listed below demonstrates the basic step-by-step process used at each monitoring node upon receiving a single update tuple. This process assumes that no recency is involved in the monitoring process.

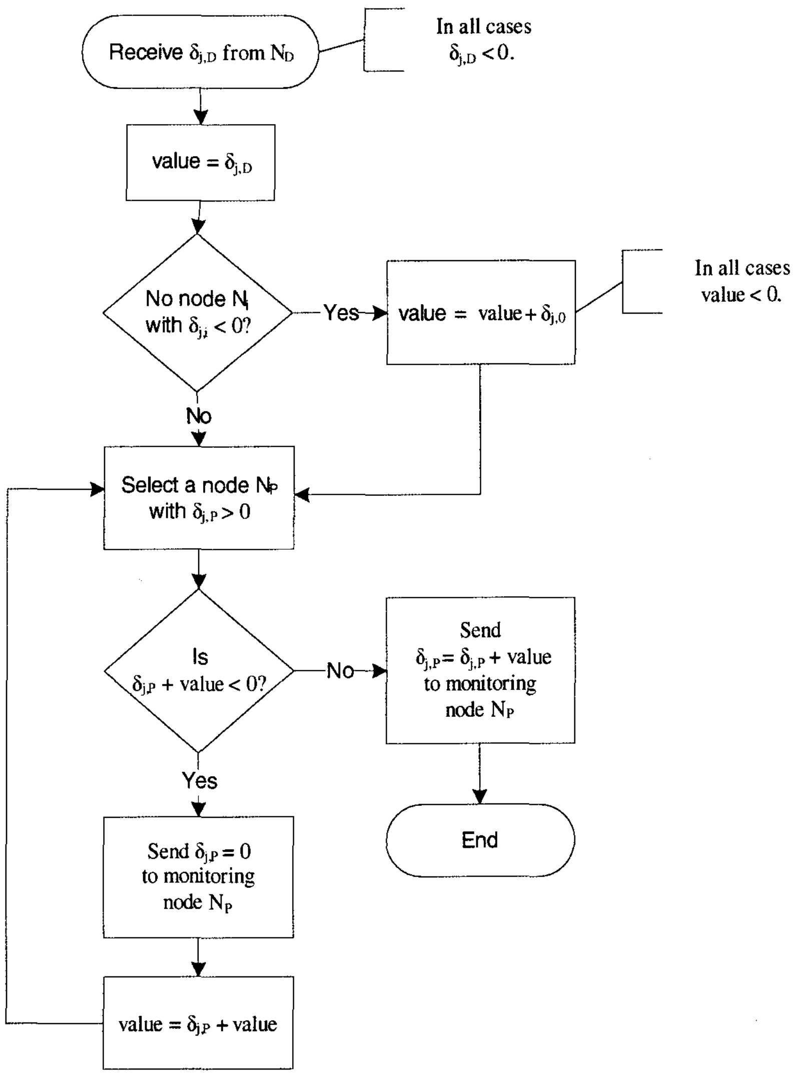


Update tuple ‹o_j, t_j› received
$T_i = T_i + s$
Is counter $c_{j,i}$ stored in memory?
No
Manage new counter
Yes
$c_{j,i} = c_{j,i} + 1$
Are all constraints valid?
No
Send invalid constraints to coordinator
Yes
End

APPENDIX II

COORDINATOR FORWARDING PROCESS

The flow chart below demonstrates the step-by-step process used at the coordinator node upon receiving a forwarded adjustment factor (see Section 3.3.3).


Receive $\delta_{j,D}$ from N_D
In all cases $\delta_{j,D} < 0$.
value = $\delta_{j,D}$
No node N_i with $\delta_{j,i} < 0$?
Yes
value = value + $\delta_{j,0}$
In all cases value < 0.
No
Select a node N_P with $\delta_{j,P} > 0$
Is $\delta_{j,P}$ + value < 0?
No
Send $\delta_{j,P} = \delta_{j,P}$ + value to monitoring node N_P
End
Yes
Send $\delta_{j,P} = 0$ to monitoring node N_P
value = $\delta_{j,P}$ + value

APPENDIX III

DATA SET DETAILS & EXAMPLES

In Section 4.1 we described the two data sets used in our evaluation experiments. The first data set described was the Lawrence Berkeley TCP data set. Example data was given in Table I. The second data set used was the 1998 World Cup data set. Below we show example data from this data set. We only show certain fields due to page size constraints.

Client ID	Date	Status	Size	Server	Object ID
1051164	09/Jun/1998:16:30:21	200	1699	13	138
1227767	09/Jun/1998:16:30:21	200	4754	16	10371
16217	09/Jun/1998:16:30:21	200	348	16	25195
350507	09/Jun/1998:16:30:21	200	870	16	59
988304	09/Jun/1998:16:30:21	200	106	16	24668
915299	09/Jun/1998:16:30:21	200	19686	16	1687
674830	09/Jun/1998:16:30:21	200	665	16	218
1227743	09/Jun/1998:16:30:21	200	665	16	218
12905	09/Jun/1998:16:30:21	200	14432	16	8
978141	09/Jun/1998:16:30:21	200	472	16	24677
932973	09/Jun/1998:16:30:21	200	1077	20	6149
890039	09/Jun/1998:16:30:21	200	4032	20	13779
1255525	09/Jun/1998:16:30:21	200	498	20	86
608386	09/Jun/1998:16:30:21	200	5024	20	13798
1255497	09/Jun/1998:16:30:21	200	1105	21	174

Finally, in Section 5.3, we described the data set used to evaluate our monitoring approach as a possible DDoS detection tool. The data set we used was a combination of normal UDP traffic traces with four different types of DDoS attack traces. All traces used are publicly available at http://lever.cs.ucla.edu.

The normal traffic traces used in our combined data set are listed below with their relative URL. The final timestamp recorded in file8 was listed at 21153 seconds. Thus the combined data sets represent less than six hours of UDP normal traffic.

file1	/ddos/traces/public/trace8/udp/file1
file2	/ddos/traces/public/trace8/udp/file2
file3	/ddos/traces/public/trace8/udp/file3
file4	/ddos/traces/public/trace8/udp/file4
file5	/ddos/traces/public/trace8/udp/file5
file6	/ddos/traces/public/trace8/udp/file6
file7	/ddos/traces/public/trace8/udp/file7
file8	/ddos/traces/public/trace8/udp/file8

The constant rate attack was inserted within file2 normal traffic trace, at approximately 2000 seconds. Since the attack trace started at zero seconds, we renumbered all timestamps relative to the new start time. The constant rate attack traces used are listed below with their relative URL.

file1	/ddos/traces/public/usc/trace3/exp1/udp/file1
file2	/ddos/traces/public/usc/trace3/exp1/udp/file2

The increasing rate attack was inserted within file4 normal traffic trace, at approximately 7000 seconds. Again, since the attack trace started at zero seconds, we renumbered all timestamps relative to the new start time. The increasing rate attack traces used are listed below with their relative URL.

file1	/ddos/traces/public/usc/trace3/exp3/udp/file1
file2	/ddos/traces/public/usc/trace3/exp3/udp/file2

The pulsing rate attack was inserted within file6 normal traffic trace, at approximately 12000 seconds. The pulsing rate attack trace used is listed below with its relative URL.

file1	/ddos/traces/public/usc/trace3/exp2/udp/file1

Finally, the gradual pulse rate attack was inserted within file8 normal traffic trace, at approximately 17000 seconds. The gradual pulse rate attack trace used is listed below with its relative URL.

file1	/ddos/traces/public/usc/trace3/exp4/udp/file1

APPENDIX IV

REALLOCATION CORRECTNESS PROOFS

In this section the reallocation method introduced in Section 3.2.2 is proven to assign adjustment factors meeting each of the three requirements given. Recall that the first adjustment factor requirement calls for the $\sum_{0 \le i \le m} \delta_{j,i} = 0$. The following theorem shows that the reallocation method used meets this requirement.

THEOREM 5. Given the reallocation method introduced in Section 3.2.2, when the method terminates $\sum_{0 \le i \le m} \delta_{j,i} = 0$.

PROOF. Theorem 5 can be proven in a very similar fashion as the correctness proof provided by Babcock and Olston (2003). First consider that before adjustment factors are assigned, it is assumed each $\delta_{j,i} = 0$. The theorem clearly holds in this initial case. Next consider when a set of participating nodes η are assigned new adjustment factors $\delta'_{j,i}$. The sum of adjustment factors in this case is not altered:

$$\sum_{i \in \eta} \delta'_{j,i} = \sum_{i \in \eta}(T_i - c_{j,i} + F_i \cdot \Delta_j)$$

$$= \sum_{i \in \eta} T_i - \sum_{i \in \eta} c_{j,i} + \Delta_j$$

$$= \sum_{i \in \eta} T_i - \sum_{i \in \eta} c_{j,i} + (\sum_{i \in \eta} c_{j,i} - \sum_{i \in \eta} T_i + \sum_{i \in \eta} \delta_{j,i})$$

$$= \sum_{i \in \eta} \delta_{j,i}$$

Since the sum is never altered from the initial base case, the reallocation method introduced in Section 3.2.2 meets the first adjustment factor requirement and terminates with $\sum_{0 \leq i \leq m} \delta_{j,i} = 0$. □

The final two adjustment factor requirements are very similar in nature and are thus proven in the same fashion. Therefore, only a formal proof for adjustment factor requirement 2 is given. This requirement calls for $\delta_{j,0} \geq 0$ for each item $o_F \in F$. The following theorem shows that the reallocation method used meets this requirement.

THEOREM 6. Given the reallocation method introduced in Section 3.2.2, when the method terminates for each item $o_F \in F$ its corresponding adjustment factor $\delta_{F,0} \geq 0$.

PROOF. Let η be a set of participating nodes involved in the reallocation process. Since the item $o_F \in F$, $\sum_{i \in \eta} c_{F,i} + \sum_{i \in \eta} \delta_{F,i} \geq \sum_{i \in \eta} T_i$. This is true for the following two reasons:

1. If $\eta = \{N_0, N_1, \ldots, N_m\}$, then the above statement is true by the membership definition given in Section 1.2.
2. If $\eta = \{N_I, N_0\}$ then the reallocation process was initiated following Phase 2 of resolution. For this to occur the above statement must be true otherwise validation testing (Section 3.2.1) would have failed and reallocation would not have been initiated.

Given the above property, $\Delta_F = \sum_{i \in \eta} c_{F,i} + \sum_{i \in \eta} \delta_{F,i} - \sum_{i \in \eta} T_i \geq 0$. Allocating a portion of this value results in $\delta_{F,0} \geq F_0 \cdot \Delta_F \geq 0$ since $F_0 \geq 0$. Thus the reallocation method introduced in Section 3.2.2 meets the second adjustment factor requirement and terminates with $\delta_{F,0} \geq 0$. □

CURRICULUM VITAE

NAME: Robert Harrison Fuller

ADDRESS: Department of Computer Engineering and Computer Science
University of Louisville
Louisville, KY 40292

DOB: Louisville, KY – June 9 1984

EDUCATION
& TRAINING: B.S., Computer Science
Indiana University Southeast
2002 – 2005

AWARDS: Chancellor's List
Indiana University Southeast
Spring 2003 – Spring 2005

Computer Sciences (Math/Science) Outstanding Student Award
Indiana University Southeast
May 2005

HONOR SOCIETIES: Alpha Chi National College Honor Scholarship Society
Indiana University Southeast
February 2005

Golden Key International Honour Society
University of Louisville
September 2006