Transactions on Machine Learning
and Data Mining
Vol. 1, No. 2 (2008) 67 - 82
©ISSN:1865-6781 (Journal),
IBaI Publishing ISSN 1864-9734

ibai Publishing
www.ibai-publishing.org

# Distributed Monitoring of Frequent Items

Robert Fuller and Mehmed Kantardzic

Computer Engineering and Computer Science Department
University of Louisville, Louisville, KY 40292
{rhfull01, mmkant01}@louisville.edu

**Abstract.** Monitoring frequently occuring items is a recurring task in a variety of applications. Although a number of solutions have been proposed there has been few to address the problem in a distributed networked environment. Most past solutions relied upon approximating results to lower communication overhead. In this paper we introduce a new algorithm designed for continuously tracking frequent items over distributed data streams providing either exact or approximate answers. We tested the efficiency of our method using two real-world data sets. The results indicated significant reduction in communication cost when compared to naïve approaches and an existing efficient algorithm called Top-K Monitoring. Since our method does not rely upon approximations to reduce communication overhead and is explicitly designed for tracking frequent items, our method also shows increased quality in its tracking results.

**Keywords:** data stream, distributed data mining, frequent items, continuous query.

## 1 Introduction

Many applications require discovering items in a data stream 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. An important application of this problem is that of detecting distributed denial of services (DDoS) attacks in a network. DDoS attacks are characterized by a surge of traffic used to overload the resources of a victim [1]. Recently, methods have been used to detect these attacks by identifying destination addresses which have received a large number of packets over a given time [2–4]. Additionally, similar tracking

tasks can be used for network flow and traffic management [5], worm detection [6], and click-fraud detection [7].

In this paper we consider the problem of monitoring frequent items over distributed data sources. More precisely, given a set of data streams originating from dispersed sources, we will report the up-to-date list of frequently occurring items in real-time. This problem is difficult since it inherits the challenges that many data stream analysis tasks possess. These challenges include a very rapid rate of data entry with potentially no foreseeable end point. To overcome these obstacles past solutions have opted to process the stream in only a single pass [8]. This often provides for a faster response time needed to keep pace and prevents the need to buffer the stream into memory. Data stream analysis is made additionally more difficult when placed in a distributed environment. Not only must computation constraints be maintained, but communication must be limited to observe any imposed network constraints.

Due to the difficulties described, it is not surprising that few solutions have been proposed [9, 10, 3]. Most available solutions focus only on the computational constraints such as memory requirements, and were not designed to operate in a distributed environment [11–15]. In this paper we present the FIDS (Monitoring Frequent Items over Distributed Data Streams) system also introduced in our preliminary work [16]. Heavily influenced by an existing method we call Top-K Monitoring [9], the FIDS system is explicitly designed to track frequent items in a distributed environment. Additional contributions are made in this work by providing an approach to reducing and bounding memory requirements. Work in this direction is important in applications where memory is heavily constrained.

The remainder of this paper will be organized as follows. Section 2 gives a formal definition of our problem and discusses prior work in the defined domain. Sect. 3 we describe in detail the FIDS system. Extensions to account for memory constraints are provided in Sec. 4. We evaluate our method based on a series of criteria in Sect. 5. Finally, closing remarks are given in Sect. 6.

## 2 Preliminaries

### 2.1 System Architecture

In this paper we considered a distributed monitoring environment defined as a single-level hierarchical architecture [9]. It consists of $m$ monitoring nodes and a specialized coordinator node. Of the nodes $N_1, N_2, \ldots, N_m$ each observe and summarize a single local data stream $S_i$. Partial knowledge of these summarizes are collected at the specialized coordinator node $N_0$. With the knowledge gathered, the coordinator is responsible for reporting continuously the set of frequent items over the union of the $m$ distributed data streams. As in previous work [9, 10], communication is only conducted among the monitoring nodes and the coordinator node. No direct communication between monitoring nodes is allowed. A schematic of this architecture can be seen in Fig. 1.

The locally observed data streams at each node, $S_1, S_2, \ldots, S_m$ are modeled as a sequence of update tuples. Each update tuple is of the form $\langle o_j, t \rangle$, where

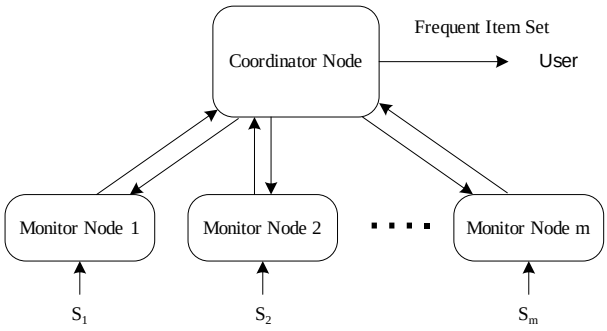


**Fig. 1.** Single-level distributed monitoring architecture.

$o_j$ is an item pulled from a (possible large) set of allowable identifiers $U$ and $t$ represents the timestamp. To summarize the data streams each monitoring node $N_i$ maintains a set of frequency counts $C_i = \{c_{1,i}, c_{2,i}, \ldots, c_{n,i}\}$. For each update tuple $\langle o_j, t\rangle$, $c_{j,i}$ is incremented by one. Therefore, each data stream is summarized as a frequency distribution set.

### 2.2 Frequent Item Problem Definition

The responsibility of the coordinator node, in our monitoring structure, is to report all items which are globally frequent. We define an item $o_j$ globally 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 $N = \sum_{1\leq i\leq m} |S_i|$. The collection of all items meeting this criterion we call the frequent item set, denoted by $F$.

To allow approximate results we adopted an extended definition, called the $\epsilon$-deficient frequent items problem [14]. In this scenario the user provides a desired error tolerance quantity controlled by variable $\epsilon \ll s$. Each counter $c_{j,i}$ on monitoring node $N_i$ is then replaced by an estimation counter $\hat{c}_{j,i}$, which at all times $c_{j,i} - \epsilon \cdot N \leq \hat{c}_{j,i} \leq c_{j,i}$. Since each counter now only provides estimated frequencies, which is potentially less than its true frequency by a bounded amount, we report all items $o_j$ as frequent if $\sum_{1\leq i\leq m} \hat{c}_{j,i} \geq (s - \epsilon) \cdot N$.

The purpose of providing approximate results is to reduce memory required on each monitoring node. This comes at a cost of reporting some false positives (items which are not truely frequent). The amount of false positives is governed both by the way the estimation counters are managed and the level of error tolerance. (Note, if $\epsilon = 0$ we will provide exact results in the fashion discussed at the beginning this section.)

### 2.3 Prior Work

Most prior work has focused primarily in the single stream problem domain. The goal of these solutions are to reduce the number of counters stored at a monitoring site. That is, they commonly address the problem of finding the $\epsilon$-deficient frequent items.

Several algorithms have been proposed to solve the $\epsilon$-deficient frequent items problem. Two popular methods are Lossy Counting and Frequent algorithms. Lossy Counting, one of two methods proposed by Manku and Motwani, requires $O(\frac{1}{\epsilon} \cdot \log(\epsilon N))$ space [14]. Frequent requires only $O(\frac{1}{\epsilon})$ space and $O(1)$ time [13], and although theoretically an improvement, Lossy Counting requires less space in practice on skewed data [11]. More recently an algorithm called Space-Saving has been proposed, which was compared against several prior solutions. This method provided significantly better precision at the cost of using more space [15].

Very little prior work has been completed in the distributed stream environment. Prior solutions can be categorized into three approaches. The first category we call a periodic approach. An example of this approach was proposed in [3]. In this paper frequency counts were propagated up a hierarchical communication structure following a fixed time period. To reduce communication, frequency precision at each level of the communication tree was addressed. A major draw back of this method is that it cannot provide continuous results without requiring a high communication overhead. The second category we call a cache approach. Cormode and Garofalakis in [10] introduced an approach of this type. Their method maintains a summary of the input stream and a prediction sketch at each monitoring node. If the summary deviants from the prediction sketch by more than a user defined tolerance amount, the summary and (possibly) a new prediction sketch is sent to a coordinator node. A major drawback of this approach is that the total error tolerance must be significant enough to prevent small deviations from triggering summary refreshes. The third category we call the synchronization approach. The Top-K Monitoring approach by Olston and Babcock is an example of this approach [9]. Their solution addressed the similar problem of finding the top-$k$ occurring items, which can can be used to monitor frequent items indirectly.

### 2.4 Our Contributions

It was the goal of this work to build off the experiences of the past in order to provide a system to explicitly monitor frequent items in a distributed stream environment. The solution was required to minimize communication overhead and consider computation constraints such as memory. In this paper we will introduce this system, examining the following three points:

1. Experimentally demonstrate the weaknesses of tracking frequent items utilizing Top-K Monitoring. The results indicate, although adequate under certain circumstances, better approaches should be considered.

2. Present a new approach which greatly reduces communication overhead and considers the problem directly for improved output quality. We call this new approach the FIDS monitoring system which is heavily influenced by Top-K Monitoring.
3. Include extensions to the FIDS system for reducing memory requirements by accommodating $\epsilon$-deficient frequent items. Evaluation of our approach indicates little overhead when compared to the worst case memory requirements of less complex single stream applications.

## 3 Frequent Item Monitoring

FIDS begins following a specified length of time called an initialization phase. This phase can be accomplished in two ways. One option is to issue an efficient one-time frequent item query. This option will reduce communication overhead, however, monitoring will not begin until the time period has elapsed. The second option is to forward all update tuples to the coordinator node. This will require more communication overhead but allow monitoring to begin immediately. Depending on the needs of the user any of these two methods can be used, although it is highly recommended that an initialization phase is used.

Once initialization is completed, the coordinator broadcasts the current global frequent item set to each monitoring node. With the current global set, each monitor then installs a series of parameterized constraints. These constraints consist of two core components and are used to determine if the global frequent item set has changed over time.

The first component of the parameterized constraints is a local threshold value $T_i$, kept by each corresponding monitoring node $N_i$. The value of each threshold is managed in a fashion so that at all times $T_i = s \cdot |S_i|$. This is achieved by incrementing $T_i$ by the user defined support $s$ for each input tuple to $N_i$. Summing the threshold values across each monitoring nodes we see that $T_\bullet = \sum_{1 \leq i \leq m} T_i = \sum_{1 \leq i \leq m} s \cdot |S_i| = s \cdot N$. Thus the global frequent item threshold is divided amongst each monitoring node, and each local threshold $T_i$ represents a portion of this division.

The second component of the parameterized constraints is a series of adjustment factors. These adjustment factors are borrowed directly from the Top-K Monitoring approach and are used to shift item occurrences amongst the nodes to facilitate local constraint checking [9]. For each item $o_j$ and node $N_i$ a corresponding adjustment factor $\delta_{j,i}$ is defined. For the correctness of the monitoring approach, each adjustment factor must meet three requirements:

1. For each item $o_j$ its corresponding adjustment factors must sum to zero across all nodes: $\sum_{0 \leq i \leq m} \delta_{j,i} = 0$.
2. For each item $o_f \in F$, its corresponding adjustment factor at the coordinator node is greater than or equal to zero: $\delta_{f,0} \geq 0$.
3. For each item $o_{nf} \notin F$, its corresponding adjustment factor at the coordinator node is less than or equal to zero: $\delta_{nf,0} \leq 0$.

Utilizing the two core components just introduced, we can now define the parameterized constraints. 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: $c_{j,i} + \delta_{j,i} \geq T_i$.
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 $o_j \in F$, $\sum_{1 \leq i \leq m} c_{j,i} + \sum_{0 \leq i \leq m} \delta_{j,i} \geq \sum_{1 \leq i \leq m} T_i$ or $\sum_{1 \leq i \leq m} c_{j,i} \geq T_\bullet$. Likewise for every $o_j \notin F$, $\sum_{1 \leq i \leq m} c_{j,i} + \sum_{0 \leq i \leq m} \delta_{j,i} < \sum_{1 \leq i \leq m} T_i$ or $\sum_{1 \leq i \leq m} c_{j,i} < T_\bullet$. Thus, as long as all constraints hold, the set of frequent items is guaranteed to be valid. In the event any one constraint is violated, the coordinator is notified that the current set may no longer be valid. At this point the coordinator begins a process called resolution to determine the new frequent item set.

### 3.1 Resolution

Whenever a local constraint is broken on any monitor node a three phase process called resolution is initiated. The purpose of this process is to determine if the frequent items set has changed and to assign new adjustment factors so that all parameterized constraints hold. This process is modified from Top-K Monitoring changing validation tests and message content. The changes made to the three phases are described below.

To begin the resolution process, in Phase 1 the monitor containing an invalid constraint $N_I$ sends a message to the coordinator. This message contains a set of frequency counts, adjustment factors, and item identifiers which are involved in violated constraints. Also included in the message sent to the coordinator, is the local threshold value of the monitor. This value is used later for calculating the new adjustment factors.

It is important to note that the entire frequent item set does not need to be sent to the coordinator. The membership of an item in $F$ is independent of any other item. As we will see later this is very important in reducing communication overhead, when comparing FIDS to Top-K Monitoring.

In Phase 2 the coordinator node determines if the frequent item set is still valid using information gathered from $N_I$ and its own stored adjustment factors. For each violated constraint, the coordinator performs the following 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 $c_{j,I} + \delta_{j,I} + \delta_{j,0} < T_I$.

In the event that all violated constraints passed their respective tests, a process called reallocation is initiated and resolution terminates. If any one test fails, however, Phase 3 is initiated instead. In Phase 3 of resolution, the coordinator contacts each monitoring node $N_i : i \neq I$ and collects the frequency counts, adjustment factors, and item identifiers corresponding to those involved in violated constraints on $N_I$. Also collected, are the local threshold values for each monitor contacted. Once all the values are collected the new frequent item

set is determined, reallocation is initiated, and resolution terminates. Phase 3 of resolution can also be called a synchronization phase, as all monitors in the network are contacted to determine the new set $F$.

### 3.2 Reallocation

Before the resolution process can terminate, new adjustment factors must be assigned to all nodes involved in the resolution process. Borrowing from Top-K Monitoring, we call this set of nodes $\mathcal{N}$ the participating nodes. If resolution terminated after Phase 2, then $\mathcal{N} = \{N_I, N_0\}$, otherwise, $\mathcal{N} = \{N_0, N_1, \ldots, N_m\}$ [9]. The process responsible for all 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. That is, we sum all weighted frequencies $(c_{j,i} + \delta_{j,i})$ and sum all thresholds retrieved from the participating nodes. Next the difference $\Delta_j$ is calculated by subtracting the two sums respectfully. Whenever $\mathcal{N}$ contains all the monitoring nodes, $\Delta_j$ represents the amount an item is over or under the global threshold.

The third step of reallocation is the tightening process. For each monitoring node $N_i \in \mathcal{N}$ and item $o_j$, we assign a new adjustment factor $\delta'_{j,i}$ so that the adjusted frequency is equal to the local threshold value. Doing this step alone is enough to guarantee that each item in $F$ will have valid constraints.

The final and fourth step, assigns a portion of $\Delta_j$ to the new adjustment factor assigned in Step 3. The amount added is based on an allocation parameter $0 \leq F_i < 1$ corresponding to node $N_i$. Allocation parameters are set in a fashion to control the amount of $\Delta_j$ given to node $N_i$ and is required that $\sum_{0 \leq i \leq m} F_i = 1$. This notation is similar to that of Top-K Monitoring with exception that $F_i \neq 1$. That is, we can not assign $\Delta_j$ entirely to any single 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 previously, we must assign a portion of $\Delta_j$ 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 two expressions.

1. $\Delta_j = \sum_{i \in \mathcal{N}} c_{j,i} + \sum_{i \in \mathcal{N}} \delta_{j,i} - \sum_{i \in \mathcal{N}} T_i$.
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 the final steps. For each item involved in an invalid constraint $o_j$, both expressions are evaluated to determine the new adjustment factor $\delta'_{j,i}$ where $i \in \mathcal{N}$ represents node $N_i$. Comparing these two equations to those used in Top-K Monitoring will show that the reallocation method original designed can be re-used. Assigning the parameters used in Top-K Monitoring appropriately will result in the definitions given above.

# 4 Monitoring Memory Reduction

## 4.1 Frequency Count Reduction

To reduce memory requirements for our method we *cannot* store a frequency count and corresponding adjustment factor for each observed item. Instead we can only store a subset of observed items utilizing techniques described in Sect. 2.3. Each of these solutions has the commonality of reducing space by utilizing estimation counters.

In this paper we adopted the MG algorithm to manage the frequency counts on each monitoring node [13, 17, 18]. This algorithm requires only $O(\frac{1}{\epsilon})$ counters to summarize a data stream. The MG algorithm works by maintaining a set of counters for each item observed, decrementing all counters by one when there are more than $\frac{1}{\epsilon}$ counters in memory. Any counter with a frequency of zero does not need to be stored and is thus removed from memory. With this method it is easy to prove that each local counter is under counted by at most $\epsilon \cdot |S_i|$ and thus comply with our definition of estimation counters.

Since the MG algorithm under-counts each frequency, we must modify our local thresholds $T_1, T_2, \ldots, T_m$. There are two approaches of making this modification. First, for each update tuple observed at node $N_i$ we can increment $T_i$ by $(s - \epsilon)$. Maintaining the threshold value in this fashion we see that $T_\bullet = \sum_{1 \leq i \leq m} T_i = \sum_{1 \leq i \leq m} (s-\epsilon) \cdot |S_i| = (s-\epsilon) \cdot N$, which complies with our definition of the $\epsilon$-deficient frequent items problem. Second, for each update tuple observed at node $N_i$ we increment $T_i$ by one. Whenever a batch decrement occurs, we also decrement the threshold. Since there can be at most $\epsilon \cdot |S_i|$ decrements at each node, we see that $(s - \epsilon) \cdot N \leq T_\bullet \leq s \cdot N$. We believe this second approach will greatly improve the quality of our results by providing a threshold that more accurately reflects the error injected into our counters.

## 4.2 Adjustment Factor Maintenance

Recall from the previous section, that the MG algorithm removes any counter with frequency equal to zero. However, we cannot remove an associated non-zero adjustment factor, otherwise we invalidate adjustment factor requirement 1 (see Section 2.3). To prevent this, all corresponding negative value adjustment factors are forwarded to the coordinator. Since the value of the adjustment factor is negative, this will not invalidate requirement 3. We maintain, however, all positive value adjustment factors. To prevent an accumulation of these values, the coordinator redistributes forwarded negative adjustment factors to the remaining monitoring nodes with positive adjustment factors. Theorem 2 shows that it is always the case that such a node exists.

**Theorem 1.** *If there is an adjustment factor $\delta_{j,i} < 0$ with corresponding counter $c_{j,i} = 0$ at node $N_i$, then there exists a monitoring node $N_p$ containing $\delta_{j,p} > 0$.*

*Proof.* With adjustment factor requirement 1 we know that $\sum_{1 \leq i \leq m} \delta_{j,i} = 0$. 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. 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 which nodes contain a positive adjustment factor, we can store all adjustment factor assignments at the coordinator. 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. We only want to forward enough so that to cancel the positive adjustment factor out at the receiving nodes, otherwise, we may be forced to repeat this step over wasting communication.

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 factor or during the reallocation process.

### 4.3 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 2.** *Each monitoring node uses at most $O(\frac{m}{\epsilon})$ counters and corresponding adjustment factors.*

*Proof.* Using the MG algorithm and the adjustment factor maintenance policy given, there are at most $\frac{1}{\epsilon}$ 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 we will have $\frac{m-1}{\epsilon}$ positive adjustment factors. Thus, we see that the memory requirements is $\mathrm{O}(\frac{m}{\epsilon})$. □

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

**Theorem 3.** *The coordinator node has at most $O(\frac{m^2}{\epsilon})$ 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. In this case, we have $\frac{m}{\epsilon}$ unique items in the system and $\frac{m}{\epsilon} \cdot (m+1)$ adjustment factor assignments. Thus the coordinator node stores at most $\mathrm{O}(\frac{m^2}{\epsilon})$ adjustment factors in memory. □

# 5 Experimental Evaluation

## 5.1 Data Sets and Experiment Methodology

**Data Sets** To evaluate the FIDS system we used two public data sets. The first data set consists of wide-area network traffic between Lawrence Berkeley Laboratory and the rest of the world [19]. The data set contains 1.8 million TCP packets with 1,622 unique user IDs. To simulate a distributed environment we evenly assigned each packet to one of four nodes and tracked frequent users. The second data set consists of 1998 World Cup web requests on 9th June [20]. The data set contains approximately 20 million requests with nearly 10,000 unique requested item IDs. For our monitoring task, we tracked frequently requested object IDs using 26 monitoring nodes (one for each active server).

**Performance Measures** For our experimentation we evaluated performance using two criteria. The first criteria is communication cost. In all our studies, we preformed exact counting and did not use our adjustment factor maintenance policy. As a result, communication is only conducted during resolution. The number of bits transmitted per resolution phase (EPR) can be formally expressed with the following equation:

$$\text{EPR} = 128 \cdot |\mathcal{F}| \cdot |\mathcal{N}'| + 64 \cdot |\mathcal{N}'| + 96 \cdot |\mathcal{F}| \cdot |\mathcal{N}'| \ . \tag{1}$$

In (1) we define $|\mathcal{F}|$ as the number of invalid constraints and $\mathcal{N}' = |\mathcal{N} - \{N_0\}|$. We make two assumptions on data representation. First all local threshold and adjustment factors are represented with 64 bits and all remaining elements (including each update tuple) require 32 bits.

Our second performance criteria measures output quality. From information retrieval we adopted the concepts of precision and recall. Precision is the percentage of relevant items found in the output and recall is the percentage of relevant items found compared to the total possible [12]. To measure the overall output quality we adopted the equation provided in [21] listed below.

$$\text{F-Measure} = \frac{2PR}{(P+R)} \ . \tag{2}$$

## 5.2 Experimental Results

**Parameter Effects on Communication Cost** The FIDS system consists of a number of user defined parameters. In our studies we investigated the effects of the support value $s$ and the coordinator allocation parameter $F_0$ on communication cost. Figure 2 and Fig. 3 shows the effects of these two parameters using the two data sets described in Sect. 5.1.

The figures show opposite results with respect to the effects of $F_0$. We see for the Berkeley TCP data set that as the allocation parameter is increased communication cost also increased, but the opposite occurs with the World Cup data

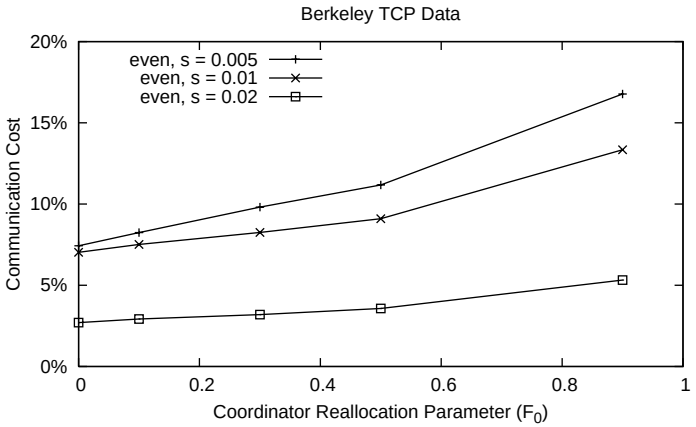


**Fig. 2.** Communication cost for Berkeley data set.

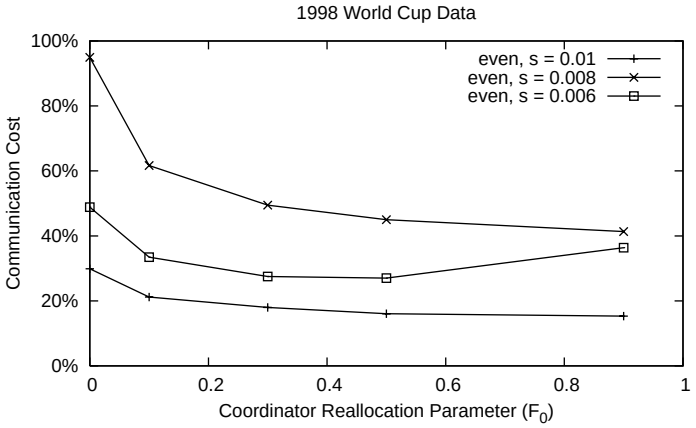


**Fig. 3.** Communication cost for '98 World Cup data set.

set. The analysis of Top-K Monitoring demonstrated that when $F_0$ is increased resolution can terminate more often in Phase 2, however, constraints are broken more frequently [9]. This same phenomenon is exhibited with our FIDS system. Since the Berkeley TCP data set consists of only four monitors, Phase 3 required little communication cost and the weaker constraints could not offset this cost. Derived from our observations, we recommend that when few monitoring nodes are within the system a small $F_0$ value ($< 0.3$) should be used.

With respect to the support parameter $s$, both figures demonstrate the same general pattern. As $s$ is increased, communication cost is decreased. An anomaly was exhibit, however, in the World Cup data set with $s = 0.008$. We speculate that in this scenario $F$ becomes more volatile, demonstrating the need for the data to maintain a degree of stability in order for significant reduction in communication cost to be realized.

Finally, we have yet to address the effects of monitor allocation parameters $F_1, F_2, \cdots, F_m$. In our studies we examined two heuristics for assigning these values. These two methods are defined as proportional allocation and even allocation and were both introduced in [9]. Our experiments showed no significant differences between the two methods, thus we used even allocation in all our experiments.

**Communication Cost with Time** We see in Fig. 4 how communication cost accumulates with time. To retrieve these results we set $F_0 = 0$ and allowed the support value to vary. In each case a sudden spike in communication cost is exhibited followed by a gradual increase. This gradual increase will continue even further past our stopping point of 500,000 updates until the data set is exhausted. These results demonstrate the need for an initialization phase. Extending the time period for initialization to cover the first 100,000 tuples will drastically reduce communication cost in our results.

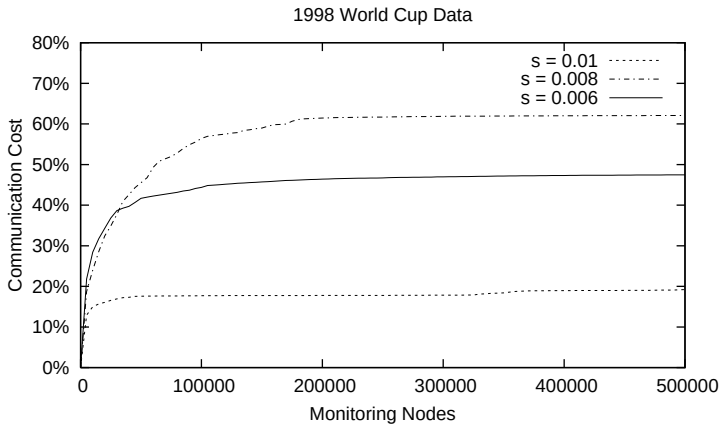


**Fig. 4.** Communication cost over time for '98 World Cup data set.

**Memory Requirements Evaluation** Our previous evaluations demonstrated communication cost with respect to exact counting. To reduce memory requirements error tolerance $\epsilon$ can be increased. We investigated the memory requirements of our policies with varying support values and settings $\epsilon = 0.1 \cdot s$. Since the majority of space required is dedicated to maintaining frequency counts, the maximum number stored on any monitoring node was used as our measure. The results of our experiments are given in Table 1.

**Table 1.** Counter statistics and worst cases.

| | Support | $O(\frac{m}{\epsilon})$ | $O(\frac{1}{\epsilon})$ | Actual Used |
|---|---|---|---|---|
| '98 World Cup | 0.010 | 26,000 | 1,000 | 1,281 |
| Data Set | 0.008 | 32,500 | 1,250 | 1,521 |
| | 0.006 | 43,334 | 1,667 | 1,895 |
| Berkeley TCP | 0.020 | 2,000 | 500 | 502 |
| Data Set | 0.010 | 4,000 | 1,000 | 1,001 |
| | 0.005 | 8,000 | 2,000 | 1,489 |

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

Also examined in our experiments was the communication cost required to maintain the adjustment factors. We saw that communication cost was between $-1.5\%$ and $0.5\%$ from our previous results. However, since all adjustment factors are stored at the coordinator node and do not need to be forwarded during resolution, communication can be reduced even more. This signifies that our adjustment factor maintenance policies are both lightweight and communication efficient.

### 5.3 Comparison

**Comparison against Top-K Monitoring** To measure the benefits gained from our FIDS system we used the results of Top-K Monitoring as a benchmark. Although Top-K Monitoring was not originally designed for monitoring frequent items, it can be used for this purpose. Assigning $K = \frac{1}{s}$ will guarantee that all frequent items are reported. However, this setting will also report a large amount of false positives. To reduce the number of false positives and improve the overall quality of our results, we considered setting $K < \frac{1}{s}$. Experiments demonstrated that if $K \approx \overline{|F|}$ we can both improve output quality and reduce communication cost.

Given the above observation, we examined the communication cost and output quality using the Berkeley TCP data set. Our results for both the FIDS system and Top-K Monitoring are summarized in Table 2 below. In our comparison table we selected the coordinator allocation parameter $F_0$ which yielded the lowest communication cost.

**Table 2.** Comparison of two approaches.

| Method | Support | Avg. Output Size | Communication Cost | 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% |
| FIDS | 0.005 | 52.25 | 7.43% | 100.00% |
| | 0.01 | 21.45 | 7.03% | 100.00% |
| | 0.02 | 7.48 | 2.70% | 100.00% |

Table 2 illustrates that in all scenarios communication cost is significantly reduced with the FIDS system. This becomes more pronounced when a lower support value is supplied. In one scenario, however, Top-K Monitoring did perform better. When $K = 5$ and $s = 0.02$ communication cost is reduced by less than a percent, but the output quality is less than optimal.

With respect to output quality, the FIDS system out-performed Top-K Monitoring in all scenarios. This is not surprising since Top-K Monitoring was not specifically designed for monitoring frequent items. It must be noted, however, that if our memory management policies are introduced, the quality of the output will also be effected. This is depended upon the amount of allowable error tolerance the user specifies.

**Comparison against Prior Work** In Sect. 3 we introduced three methods applicable to our problem domain. Of the three methods, the FIDS system is the only solution that is capable of providing exact results. Both the periodic and cache approaches relied upon approximation to reduce communication cost. Top-K Monitoring, as discussed previously, also cannot produce exact results since it was not specifically design for our task.

Despite our attempts to reduce memory requirements in Sect. 4 it is not known if the FIDS system requires less memory in practice. Compared to the cache approach provided in [10], if $m > \frac{1}{\epsilon}\log(\frac{1}{\delta})$, where $(1-\delta)$ is a probabilistic confidence, our approach will in worst case use more space on each monitoring node. Since all adjustment factors are stored at the coordinator, more memory will also be required at this location.

## 6 Conclusions

In this paper we addressed the problem of continuously monitoring frequent items in a distributed data stream environment. We analyzed a previous solution called Top-K Monitoring and determined that it can also solve this problem. However, modifications to the algorithm were made which both reduced communication cost and improved the overall quality of the output. Finally, we extended our system addressing memory constraints. Empirical studies showed that memory was reduced to near worst case quantity required of any monitoring approach.

A few issues still need to be addressed to improve upon the FIDS system. First, our experiments demonstrated the importance of an initialization phase. Alternative methods should be implemented and communication cost reevaluated. Additionally, heuristics should be determined on the length of time needed to "warm-up". Second, further research in memory reduction and adjustment factor maintenance is needed to obtain an improved theoretical bound. Finally, the scalability of the FIDS system needs to be evaluated.

## References

1. Peng, T., Leckie, C., Ramamohanarao, K.: Proactively detecting distributed denial of service attacks using source IP address monitoring. NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. (2004) 771–752
2. Akella, A., Bharambe, A., Reiter, M., Seshan, S.: Detecting DDoS attacks on ISP networks. (2003)
3. Manjhi, A., Shkapenyuk, V., Dhamdhere, K., Olston, C.: Finding (recently) frequent items in distributed data streams. In: Proc. of Int. Conf. on Data Engineering (ICDE), Washington, DC, IEEE Computer Society (2005) 767–778
4. Sekar, V., Duffield, N., Spatscheck, O., van der Merwe, J., Zhang, H.: LADS: large-scale automated DDOS detection system. In: Proc. of the Annual Conf. on USENIX '06 Annual Technical Conference, Berkeley, CA, USA, USENIX Association (2006) 171–184
5. Stanojevic, R.: (Scalable heavy-hitter identification) Also available as http://www.hamilton.ie/person/rade/ScalableHH.pdf.
6. Kim, H.A., Karp, B.: Autograph: toward automated, distributed worm signature detection. In: Proc. of the USENIX Security Symposium, Berkeley, CA, USA, USENIX Association (2004) 271–286
7. Metwally, A., Agrawal, D., Abbadi, A.E.: Using association rules for fraud detection in web advertising networks. In: Proc. of the 31st international conference on Very large data bases, VLDB Endowment (2005) 169–180
8. Zhu, Y., Shasha, D.: Statstream: statistical monitoring of thousands of data streams in real time. In: Proc.s of the 28th Intl. Conf. on Very Large Data Bases, VLDB Endowment (2002) 358–369
9. Babcock, B., Olston, C.: Distributed top-k monitoring. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, New York, ACM (2003) 28–39

10. Cormode, G., Garofalakis, M.: Sketching streams through the net: distributed approximate query tracking. In: Proc. of Int. Conf. on Very Large Data Bases (VLDB), VLDB Endowment (2005) 13–24
11. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proc. of the 23rd ACM Symposium on Principles of Database System (PODS), New York, ACM Press (2004) 286–296
12. Cormode, G., Muthukrishnan, S.: What's hot and what's not: tracking most frequent items dynamically. ACM Trans. Database Syst. **30**(1) (2005) 249–278
13. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency estimation of internet packet streams with limited space. In: Proc. of Annual European Symposium on Algorithms, London, UK, Springer-Verlag (2002) 348–360
14. Manku, G.S., Motwani, R.: Approximate frequency counts over data streams. In: Proc. of the 28th Int. Conf. on Very Large Data Bases (VLDB), VLDB Endowment (2002) 346–357
15. Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams, Springer-Verlag (2005) 398–412
16. Fuller, R., Kantardiz, M.: Fids: Monitoring frequent items over distributed data streams. Volume LNAI 4571., Heidelberg, Springer Verlag (2007) 464–478
17. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. **28**(1) (2003) 51–55
18. Misra, J., Gries, D.: Finding repeated elements. Science of Computer Programming **2**(2) (1982) 143–152
19. Paxson, V., Floyd, S.: Wide-area traffic: The failure of poisson modeling. IEEE/ACM Trans. on Networking **3**(3) (1995) 226–244
20. Arlitt, M., Jin, T.: 1998 World Cup web site access logs. Available from http://www.acm.org/sigcomm/ITA/ (1998)
21. van Rijsbergen, C.: Information Retrieval. Butterworths, London (1979)