Noname manuscript No.
(will be inserted by the editor)

Identifying frequent items in distributed data sets

Jan Sacha · Alberto Montresor

the date of receipt and acceptance should be inserted later

Abstract Many practical problems in computer science require the knowledge of the most frequently occurring items in a data set. Current state-of-the-art algorithms for frequent items discovery are either fully centralized or rely on node hierarchies which are inflexible and prone to failures in massively distributed systems. In this paper we describe a family of gossip-based algorithms that efficiently approximate the most frequent items in large-scale distributed datasets. We show, both analytically and using real-world datasets, that our algorithms are fast, highly scalable, and resilient to node failures.

1 Introduction

Many problems in various areas of computer science can be reduced to the problem of discovering the most frequent items in a data set [7, 11, 5, 8]. For example, the knowledge of the most frequently accessed data is necessary to optimize cache performance in a storage system. Similarly, the knowledge of the most frequent packet flows can be used to optimize routing and resource usage in a network. Other applications include intrusion detection (e.g., worm, fraud, or distributed denial of service attack), usage pattern detection (e.g., suggestions in a search engine), and market analysis.

Classic algorithms for finding frequent items assume that all data storage and processing is centralized [7, 11, 5, 8]. To apply these algorithms to a distributed data set, one would have to fetch all data items from the system to a single node, potentially incurring in a large communication cost. Moreover, these centralized algorithms are not easily parallelized. Even if one calculates the local item occurrence on each node in a distributed system, identifying the globally frequent items is still non-trivial because the globally frequent items might have a low local occurrence at individual nodes.

More recently, new approaches have been proposed for frequent item discovery in distributed data sets [6]. However, these approaches are either based on a master-slave model [4, 2], where a master node performs all coordination and is a potential

Jan Sacha is with Alcatel-Lucent Bell Labs, Copernicuslaan 50, 2018 Antwerp, Belgium
Alberto Montresor is with the University of Trento, via Sommarive 14, 38123, Trento, Italy

performance and reliability bottleneck, or rely on node hierarchies [13,14] which are expensive in maintenance and prone to errors in large-scale, dynamic systems.

In this paper, we tackle the problem of frequent items discovery in an entirely decentralized way. We consider three variants of the problem where the goal is to determine (i) the k most frequent items, (ii) items with absolute frequency above a given threshold, and (iii) items with relative frequency above a given threshold. We introduce a family of simple gossip-based algorithms that efficiently solve all three problems in large-scale distributed datasets.

In our approach, nodes first compute local frequencies of data items and then run an aggregation protocol to discover the globally frequent items. The main advantage of our algorithm is its very high scalability and resilience to failures. We formally prove that our algorithm is correct and analytically derive upper bounds on its completion time. We then validate our approach using real-world data traces, showing that the algorithm correctly and efficiently identifies all frequent items, even under heavy churn.

The rest of paper is organized as follows. The system model and problem statement are described in Section 2. Our algorithms are introduced in Section 3 followed by a correctness proof in Section 4. Theoretical analysis and experimental validation are described in Section 5 and Section 6, respectively. Finally, Section 7 compares our approach to the state-of-the-art algorithms and Section 8 concludes the paper.

2 System model and problem statement

We consider a networked system consisting of a large collection P of computing nodes storing a distributed multiset of data items taken from the universe I. Let $N = |P|$ be the number of nodes in the system and $m = |I|$ be the number of distinct items.

Each node p stores a *local subset* of items $I_p \subseteq I$. We define *local frequency* $F_p(i)$ as the number of copies of item i at node p. For items not included in I_p, we define $F_p(i) = 0$. We assume that nodes are able to compute local item frequencies using one of the centralized algorithms discussed in Section 7.

Further, we define the *global absolute frequency* $F(i)$ and the *global relative frequency* $\hat{F}(i)$ of item i as:

$$F(i) = \sum_{p \in P} F_p(i)$$
$$\hat{F}(i) = F(i)/M$$

where $M = \sum_{i \in I} F(i)$ is the total number of items (including copies) stored in the system.

The problem we are addressing is to discover the k *most frequent* items (MF). Let j be the k-th item in the sequence of all items ordered by decreasing global frequency. We are trying to identify items whose global frequency is larger than or equal to $F(j)$:

$$MF = \{i : F(i) \geq F(j)\}$$

Note that the size of MF could be larger than k because several items may have the same global frequency as j.

Additionally, we are addressing two related problems: to identify *absolutely frequent* items (AF) whose global absolute frequency is larger than the absolute threshold f

$$AF = \{i : F(i) \geq f\}$$

and to identify *relatively frequent* items (RF) whose global relative frequency is larger than the relative threshold ϕ

$$RF = \{i : \hat{F}(i) \geq \phi\}$$

We are focusing on optimizing the communication cost in the distributed system rather than minimizing the number of local data accesses as do the centralized algorithms. As in other gossip protocols, we assume that nodes organize themselves into a P2P overlay using an inexpensive membership maintenance protocol such as a peer sampling service [10].

3 Algorithm

For presentation purposes, we first introduce a simple yet inefficient algorithm that solves the MF problem and then describe an extended FreqMF algorithm that meets practical requirements. At the end of this section we further modify the algorithm to deal with the AF and RF problems.

3.1 Naive algorithm

The naive approach to discover the most frequent items is based on explicit approximation of global item frequencies using a well-known gossip-based averaging protocol [9]. We use the averaging protocol to estimate the average item frequency, $\frac{1}{N}\sum_p F_p(i)$, which by definition is equal to $F(i)/N$. Since N is constant, items with the highest average frequency are the most frequent items in the system.

Nodes participates in m different instances of the averaging protocol, one for each item in I. Each node p maintains a map variable est_p that associates item i known by p with an estimated average frequency $est_p[i]$. Initially, $est_p[i] = F_p(i)$ for all items i stored locally by p. For an item i that node p does not know about, we define $est_p[i] = 0$. Every node p periodically selects a random neighbor q and sends the entire content of its est_p map to q. Node q replies by sending its est_q map to p. After the exchange both nodes update their estimates in the same way: $est_p[i] = est_q[i] = \frac{1}{2}(est_p[i] + est_q[i])$.

This simple push-pull gossip protocol serves two purposes. First, it propagates at an exponential rate the information about each data item to all nodes in the system. Since nodes do not forget about items, the information about each item i spreads epidemically. Second, the protocol causes the estimate values, $est_p[i]$, to converge to the average global frequency $F(i)/N$ for each item i and every node p in the system.

Note that in the absence of failures, a gossip exchange between any two nodes p and q does not change the sum of their $est_p[i]$ and $est_q[i]$ values and thus the protocol preserves the following invariant: $\sum_{p\in P} est_p[i] = \sum_{p\in P} F_p(i) = F(i)$ for each item i. Since an exchange probabilistically reduces the variance between gossiping nodes, the $est_p[i]$ variables converge to the mean $F(i)/N$. As shown by Jelasity et al. [9], this convergence is exponentially fast. A node p can thus return the k most frequent items in the system by simply selecting the k items with the highest $est_p[i]$ score.

Algorithm 1: FreqMF algorithm executed by node p

```
upon initialization do
    Map est ← new Map() foreach i ∈ I_p do
        est_p[i] ← F_p(i);

repeat every Δ time units
    Node q ← random(P);
    send ⟨request, extract(est, s)⟩ to q;

upon receive⟨request, req_q⟩ from q do
    Map rep_p ← new Map() foreach i ∈ req_q do
        real δ_i ← ½(req_q[i] − est_p[i]);
        est_p[i] ← est_p[i] + δ_i;
        rep_p[i] ← δ_i
    send ⟨reply, rep_p⟩ to q ;

upon receive⟨reply, rep_q⟩ do
    foreach i ∈ rep_q do
        est_p[i] ← est_p[i] − rep_q[i];

function getTop (int k)
    return extract(est, k);
```

3.2 FreqMF protocol

The naive protocol allows a very quick convergence but has a very high communication cost, as it requires each node p to send the entire content of its est_p map in every gossip message. It also unrealistically assumes that message exchanges are atomic and ignores message latency.

In order to address these shortcomings, we modify the protocol so that nodes exchange only subsets of their $est_p[i]$ estimates. To solve the *MF* problem, a node sends in an exchange message the s most frequent items stored in est_p, where s is a protocol parameter such that $s \geq k$. Since messages are generated periodically and have a fixed maximum size, this protocol has a constant communication cost.

We also modify the protocol to allow asynchronous message passing. The pseudocode for our modified protocol, which we call FreqMF, is shown as Algorithm 1. After initializing est_p, each node p periodically sends a request message to a randomly chosen neighbor q. The message contains the node's s highest $est_p[i]$ values selected using an extract() function. Neighbor q updates its estimates by subtracting $\delta_i = \frac{1}{2}(est_q[i] - est_p[i])$ from each corresponding $est_p[i]$ variable. Thus, $est_q[i]$ becomes $est_q[i] - \delta_i = \frac{1}{2}(est_q[i] + est_p[i])$ as in the initial naive protocol. The sum $\sum_{p \in P} est_p[i]$ of estimates over all nodes is temporarily reduced by δ_i because node p has not changed its $est_p[i]$ yet. However, q sends a reply message to p containing δ_i so that p eventually updates its $est_p[i]$ variable to $est_p[i] + \delta_i = \frac{1}{2}(est_q[i] + est_p[i])$ preserving the invariant as in the naive protocol.

The gossip exchanges reduce the variance between the $est_p[i]$ estimates at the same exponential rate as in the naive protocol. However, due to the asynchronous message exchanges, FreqMF can handle message delay and message reordering.

Interestingly, even though nodes exchange only s values per gossip cycle, all nodes in the system eventually identify all of the most frequent items in the data set. The key

observation is that nodes exchange information about all the items that are suspected of having high global occurrence, i.e., nodes that have high $est_p[i]$ scores. If an item is wrongly classified, the averaging protocol reduces its high $est_p[i]$ score towards a low $F(i)/N$ mean value. Conversely, if an item has a high global frequency, its $est_p[i]$ value remains high during exchanges and the gossip protocol spreads it to all nodes. Eventually, the system reaches a stable configuration in which the s most frequent items in the data set have the highest $est_p[i]$ scores on all nodes in the system and are thus included in every exchange message. At that point, all nodes correctly identify all the k most frequent items given that $s \geq k$. By running the protocol further, nodes exponentially decrease the variance between the $est_p[i]$ estimates and hence improve the approximation accuracy of $F(i)/N$.

3.3 FreqAF and FreqRF protocols

We have assumed so far that the *MF* problem requires identifying the k most frequent items without returning their actual global frequencies $F(i)$. However, $F(i)$ can be easily estimated using our algorithm by simply computing $est_p[i] \cdot N$, where N is the system size. If N is not known a priori, it can be easily estimated by the same averaging protocol we have used so far [9].

With this information at hand, the *AF* problem can be solved straightforwardly: instead of selecting the top s items from the *est* map, the $\mathsf{extract}(est, f)$ function returns items whose estimated global frequency $est[i] \cdot N$ is larger than the absolute threshold f. In this algorithm, items outside the *AF* set eventually fall below the f threshold and nodes exchange only items that belong to *AF*.

The *RF* problem can be solved in a similar way. We add a special item $\perp$ to the *est* map initialized to $\sum_{i \in I_p} F_p(i)$, i.e. the total number of items stored by node p. Item $\perp$ is treated as all other items and $est[\perp]$ converges to M/N, i.e., the average number of items per node, and $\frac{Est[i]}{est[\perp]}$ converges to $\frac{F(i)/N}{M/N} = \frac{F(i)}{M}$, which is equal to the relative frequency $\hat{F}(i)$. Further, the $\mathsf{extract}(est, \phi)$ function returns only the items whose estimated relative frequency $est[i]/est[\perp]$ is larger than the relative threshold ϕ.

3.4 Termination condition

The protocols defined so far run indefinitely, continuously improving the $F(i)/N$ estimation accuracy. At some point the accuracy is high enough that the item order does not change anymore and all nodes obtain exactly the same sets of most frequent items. To detect this condition and terminate the algorithm we use the following criterion. Every node records its target set (*MF*, *AF* or *RF*) at the end of each round. If the target set does not change for d consecutive rounds, where d is an algorithm parameter called *delay*, the node stops initiating exchanges and only replies to incoming requests. As we show later, using this termination condition all nodes eventually stop generating messages.

4 Convergence proof

In this section, we show that the FreqMF algorithm eventually converges and identifies the k most frequent items correctly. We say that the protocol *converges* when the frequency estimate associated with any item $i \in MF$ is larger than the frequency estimate associated with any item $j \notin MF$:

$$\forall p \in P, \forall i \in MF, \forall j \notin MF : est_p[i] > est_p[j]$$

Note that the termination condition described in Section 3.4 is a heuristic that allows the algorithm to stop but does not strictly guarantee obtaining correct results. In this section we assume the algorithm runs indefinitely as we show that it *eventually* converges.

We also assume that the MF set is well-defined, meaning that it contains exactly k items. In the case where MF is not well-defined, the problem of identifying the k most frequent items is actually simpler because multiple subsets of MF may actually be accepted as valid top-k items.

Further, note that in the absence of failures and after each protocol round the global frequency of any item i is always equal to the sum of the corresponding est variables:

$$F(i) = \sum_{p \in P} est_p[i]$$

This invariant holds because (i) variable est_p is initialized with the local frequencies of all items stored at each node p and (ii) item exchanges do not change the sum of the $est_p[i]$ variables (but do reduce the variance).

We prove the algorithm's correctness by induction on k.

Base case: $k = 1$. Let us assume that the message size s is 1. If the message size is larger it is easy to show that the algorithm converges at least as fast. Let i_1 be the most frequent item. We define $\underline{min}(i_1)$ as the minimum frequency estimate of i_1 over all nodes:

$$\underline{min}(i_1) = \min\{est_p[i_1] : p \in P\}$$

Further, we define $\overline{max}(i_1)$ as the maximum frequency estimate over all nodes and all items except i_1:

$$\overline{max}(i_1) = \max\{est_p[i] : i \neq i_1 \wedge p \in P\}$$

The protocol converges when $\underline{min}(i_1) > \overline{max}(i_1)$. In this configuration every node classifies i_1 as the most frequent item. Moreover, this configuration is stable because $\overline{max}(i_1)$ is non-increasing and $\underline{min}(i_1)$ is non-decreasing. Once $\underline{min}(i_1) > \overline{max}(i_1)$ the system cannot go back to a previous configuration where $\underline{min}(i_1) < \overline{max}(i_1)$.

Suppose that $\underline{min}(i_1) < \overline{max}(i_1)$ and so the protocol has not converged yet. A node p exchanges an item i_p that has currently the highest local frequency. There are two possible cases:

- $i_p \neq i_1$: in this case $\overline{max}(x)$ is probabilistically reduced
- $i_p = i_1$: in this case $\underline{min}(x)$ is probabilistically increased

Due to the exchanges, $\underline{min}(i_1)$ converges towards $F(i_1)/N$ and $\overline{max}(i_1)$ converges towards $F(i_2)/N$, where i_2 is the second most frequent item in the system, and thus eventually $\underline{min}(i_1)$ must be greater than $\overline{max}(i_1)$ and the algorithm must converge. In the corner case where $F(i_1) = F(i_2)$ nodes might switch between selecting i_1 and

i_2 but since both these items belong to MF the algorithm always generates correct output.

Inductive step: $k > 1$. Suppose that the protocol has already identified the most frequent $k-1$ items. Let us assume that messages have size $s = k$. Again, if message size is higher than k the algorithm converges at least as fast. Let i_k be the item of rank k and let $\underline{min}(i_k)$ and $\overline{max}(i_k)$ be equal to:

$$\underline{min}(i_k) = \min\{est_p[i_k] : p \in P\}$$
$$\overline{max}(i_k) = \max\{est_p[i] : i \notin MF \wedge p \in P\}$$

With a reasoning similar to the base case, item exchanges can only increase $\underline{min}(i_k)$ towards $F(i_k)/N$ or decrease $\overline{max}(i_k)$ towards $F(i_{k+1})/N$ where i_{k+1} is the most frequent item outside the MF set. Eventually $\underline{min}(i_k) > \overline{max}(i_k)$ and the algorithm converges.

We omit here similar proofs for FreqAF and FreqRF; it sufficient to observe that eventually all nodes will discover all items included in AF or RF, and the average estimate for all those items will eventually grow larger than the average estimate of the items not belonging to the target set.

5 Analysis

We have shown that the algorithm eventually converges and generates correct results. In this section we provide bounds on the *convergence time*, i.e. the number of rounds needed to reach convergence, in the naive algorithm. In the following section we evaluate the full FreqMF algorithm.

To discuss convergence time, we consider two orthogonal distributions: the distribution of global frequencies among all items, and the distribution of local frequencies for each item among all nodes. We first derive a generic convergence time formula valid for all distributions and then further analyze a specific scenario where global frequencies follow a Zipf distribution and items are assigned to nodes uniform at random.

5.1 General case

Let us consider, for a given item i, the distribution of local frequencies $F_p(i)$ among all nodes in the system. Let $\mu_i = F(i)/N$ and σ_i^2 be the mean and the variance of such a distribution, respectively. Let us define the *offset* δ_i of the distribution as the maximum absolute distance between one value of such distribution and the mean, so that all $F_p(i)$ values are in the range $[\mu_i - \delta_i, \mu_i + \delta_i]$.

Given a variance σ_i^2, the *maximum potential offset* Δ_i is the maximum offset that can be obtained by any distribution having such variance. It is easy to prove (by contradiction) that the maximum potential offset occurs when the local frequency of item i is equal to $\mu_i \pm \Delta_i$ in one node, while all other nodes have local frequencies equal to $\mu_i \mp \frac{\Delta_i}{N-1}$. In such case, Δ_i can be calculated as follows:

$$\begin{aligned}\sigma_i^2 &= \frac{1}{N}\left((\mu_i - (\mu_i - \Delta_i))^2 + (N-1)\left(\mu_i - \left(\mu_i - \frac{\Delta_i}{N-1}\right)\right)^2\right) \\ &= \frac{1}{N}\left(\Delta_i^2 + \frac{\Delta_i^2}{N-1}\right) = \frac{\Delta_i^2}{N-1}\end{aligned}$$

from which we obtain $\Delta_i = \sqrt{\sigma_i^2 \cdot (N-1)}$.

In the averaging protocol, the variance is expected to be reduced at each execution round by a factor $\rho = \frac{1}{2\sqrt{e}}$ [9]. Hence, the maximum potential offset is expected to be reduced accordingly. Let $\sigma_i^2(t)$ and $\Delta_i(t)$ be the variance and the maximum potential offset at round t, respectively. The offset is expected to decrease at the following rate:

$$\frac{\Delta_i(t)}{\Delta_i(t-1)} = \frac{\sqrt{\sigma_i^2(t) \cdot (N-1)}}{\sqrt{\sigma_i^2(t-1) \cdot (N-1)}} = \frac{\sqrt{\rho \cdot \sigma_i^2(t-1) \cdot (N-1)}}{\sqrt{\sigma_i^2(t-1) \cdot (N-1)}} = \sqrt{\rho} = \sqrt{\frac{1}{2\sqrt{e}}} \quad (1)$$

We are now in a position to provide a probabilistic upper bound on the convergence time. Let x be the item in MF such that $\mu_x - \Delta_x(0)$ is minimum, and let y be the item outside MF such that $\mu_y + \Delta_y(0)$ is maximum:

$$x = \operatorname*{argmin}_{i \in MF} \{\mu_i - \Delta_i(0)\}$$
$$y = \operatorname*{argmax}_{i \notin MF} \{\mu_i + \Delta_i(0)\}$$

We expect the protocol to converge when the estimated frequency of item x is greater than the estimated frequency of item y:

$$\mu_x - (\sqrt{\rho})^t \cdot \Delta_x(0) > \mu_y + (\sqrt{\rho})^t \cdot \Delta_y(0)$$

from which we can obtain the following bound on t:

$$t > 2 \log_{1/\rho} \frac{\Delta_x(0) + \Delta_x(0)}{\mu_x - \mu_y} \quad (2)$$

5.2 Specific scenario

To show how Formula 2 can be practically applied, let us now consider an example scenario in which item frequencies follow Zipf's law with skewness[1] $s = 1$ and items are distributed among nodes uniformly at random. In such a system, the frequency of item ranked k is expected to be:

$$f(k) = M \frac{1/k}{\sum_{i=1}^{m} 1/i}$$

where m is the number of distinct items and M is the total number of items in the system. Note that $f(k+1)$ can be expressed as $f(k) \cdot k/(k+1)$.

The number of k-ranked items assigned to a node follows a binomial distribution with mean $\mu(k) = f(k)/N$ and variance $\sigma^2(k) = f(k)/N \cdot (1-1/N) = f(k) \cdot (N-1)/N^2$. In fact, this scenario can be interpreted as a balls-into-bins problem [16], where $f(k)$ balls (items ranked k) are distributed among N bins (corresponding to nodes).

Zipf distributions have been shown to accurately model item distributions in many large datasets [13]. Our generic approach can be applied to other item distributions too, such as normal and uniform. In this section we focus on a Zipf distribution as the most representative case for large datasets.

[1] Other cases with $s \neq 1$ are similar.

5.3 Convergence time

The convergence time, as defined by Formula 2, is maximized when $x \in MF$ is an item ranked k and $y \notin MF$ is an item ranked $k+1$. In the worst possible case, all items are assigned to a single node so that $\mu_x - \Delta_x$ is equal to zero and $\mu_y + \Delta_y$ reaches a maximum possible value of $\mu_y + F(y)$. Hence:

$$\Delta_x = f(k)/N$$
$$\Delta_y = f(k+1) - f(k+1)/N.$$

By substituting Δ_x, Δ_y, μ_x, and μ_y in Formula 2 we thus obtain a bound on the algorithm's expected convergence time:

$$\begin{aligned} t > 2\log_{1/\rho} \frac{f(k)/N + f(k+1) - f(k+1)/N}{f(k)/N - f(k+1)/N} \\ = 2\log_{1/\rho}(1 + \frac{Nf(k+1)}{f(k) - f(k+1)}) \\ = 2\log_{1/\rho}(1 + \frac{Nf(k)\frac{k}{k+1}}{f(k) - f(k)\frac{k}{k+1}}) \\ = 2\log_{1/\rho}(1 + \frac{N\frac{k}{k+1}}{1 - \frac{k}{k+1}}) \\ = 2\log_{1/\rho}(1 + Nk) \end{aligned}$$

We validate our theoretical model by running the naive algorithm in the event-driven engine of the peer-to-peer simulator PEERSIM [15]. We run the algorithm 250 times with $M = 10^6$ and $m = 10^3$, for different values of N and we record the convergence time. Figure 1 shows the theoretical bound and the empirical results. The outputs from experiments are shifted by small random values along both axes to indicate variance. The theoretical curves are extremely low – a few rounds are sufficient to reach convergence – and yet this is only a theoretical upper bound; the naive algorithm performs much better in all simulation runs than the theoretical bound.

6 FREQMF evaluation

The theoretical analysis described in the previous section provides us with a good understanding of the naive algorithm. However, already for this simple protocol, the performance model is surprisingly complicated and our theoretical convergence bound significantly overestimates empirical results. To evaluate the the full FREQMF algorithm we thus resort to simulation experiments. In particular, we address in this section the following questions:

- How does the message size s impact on the convergence speed?
- What is the impact of the termination condition on the results?
- What is the behavior of our protocols with realistic datasets?
- How does the protocol behave in the presence of failures?

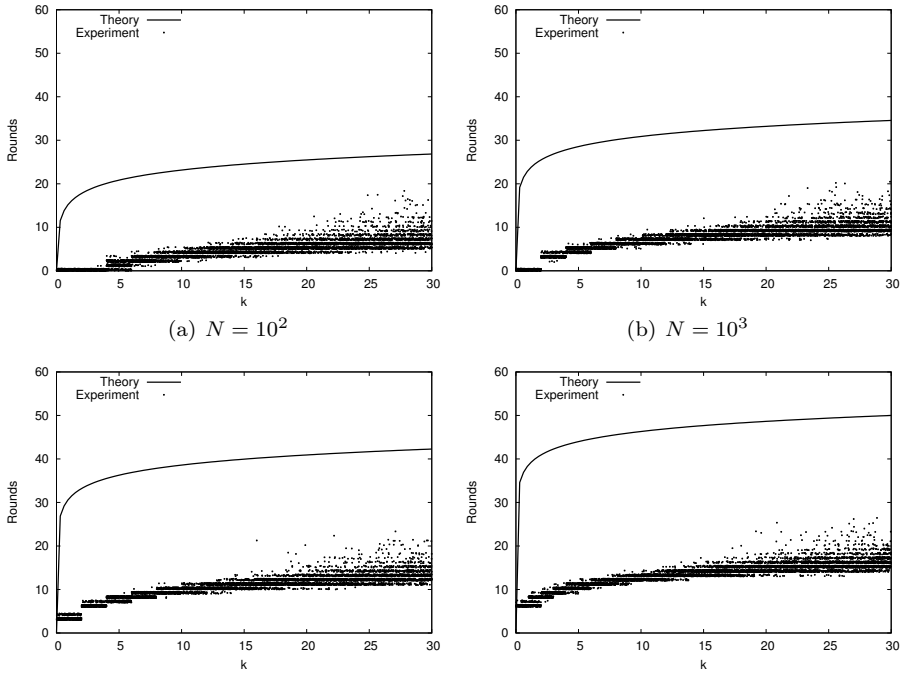


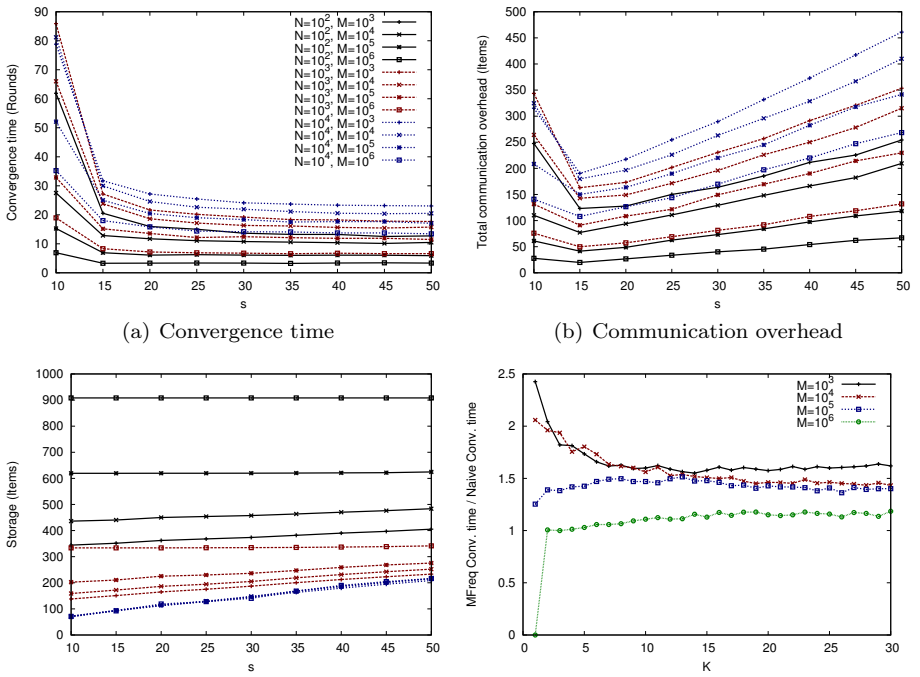
Fig. 1 Convergence time bound (curves without dots) and simulation results (individual dots) for different values of N and increasing values of k.

We evaluate FreqMF and FreqRF based on the communication overhead, storage cost, convergence speed, and result accuracy. *Communication overhead* is measured as the number of items exchanged by a node, either per round or during an experiment. *Storage cost* is the number of items stored by node during an experiment. *Convergence speed* is measured either as *convergence time*, i.e., the number of rounds needed for all nodes to correctly identify the most frequent item set, or as *execution time*, i.e., the number of rounds by which all nodes terminate the algorithm. Finally, we evaluate the result accuracy by measuring the *error* defined as the number of items in the result set that do not belong to MF.

The evaluation is performed using Peersim. Unless specified otherwise, the parameters and their default values used in this section are listed in Table 1. Each experiment is repeated 100 times and when possible each individual measurement is shown or error bars are indicated to illustrate the variability in experiments.

Parameter	**Value**	**Description**
k	10	Number of most frequent items to be selected
s	$2k$	Number of items per FreqMF messages
d	8	Termination condition

Table 1 Configuration parameters.



(a) Convergence time

(b) Communication overhead

(c) Storage cost

(d) Convergence time penalty for FreqMF over the naive protocol

Fig. 2 Behavior of the FreqMF protocol when varying parameter s. The legend of (b) and (c) are the same as (a).

6.1 Message size

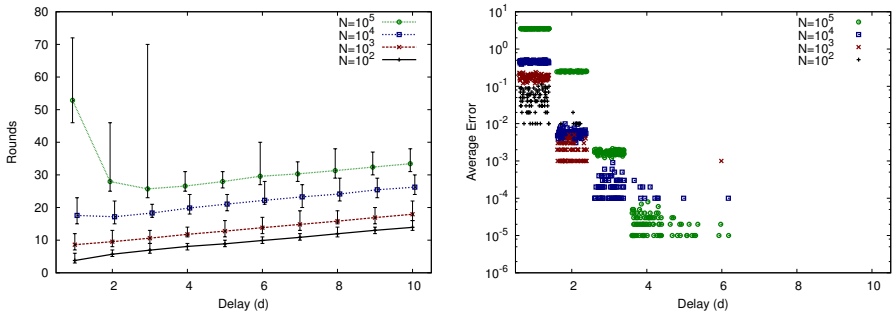
Figures 2(a), 2(b) and 2(c) show the influence of the message size s on the convergence time, communication overhead, and storage cost in FreqMF. The following experiments are based on a Zipf distribution for item frequency with skewness 1, with items distributed uniform at random between nodes. We consider several different combinations of N and M, with m set to 1000.

It is possible to observe only a marginal decrease in convergence time when the ratio s/k is larger than 2, while the overhead grows linearly after that threshold. Moreover, nearly constant storage costs show that very little additional knowledge is gained when s increases. For these reasons, we fix s at $2k$ in the remaining experiments in this paper.

Figure 2(d) compares the convergence time of FreqMF and the naive algorithm for increasing values of k. Interestingly, the ratio between FreqMF and the naive algorithm is approximately constant. FreqMF is only about 50% slower than the naive algorithm with unlimited message size.

6.2 Termination condition

With a message size fixed at $2k$ we are in a position to evaluate the termination condition explained in Section 3.4. As we see in Figure 3(a), the execution time in



(a) Execution time (b) Average error

Fig. 3 Execution time and result accuracy when the termination condition is applied.

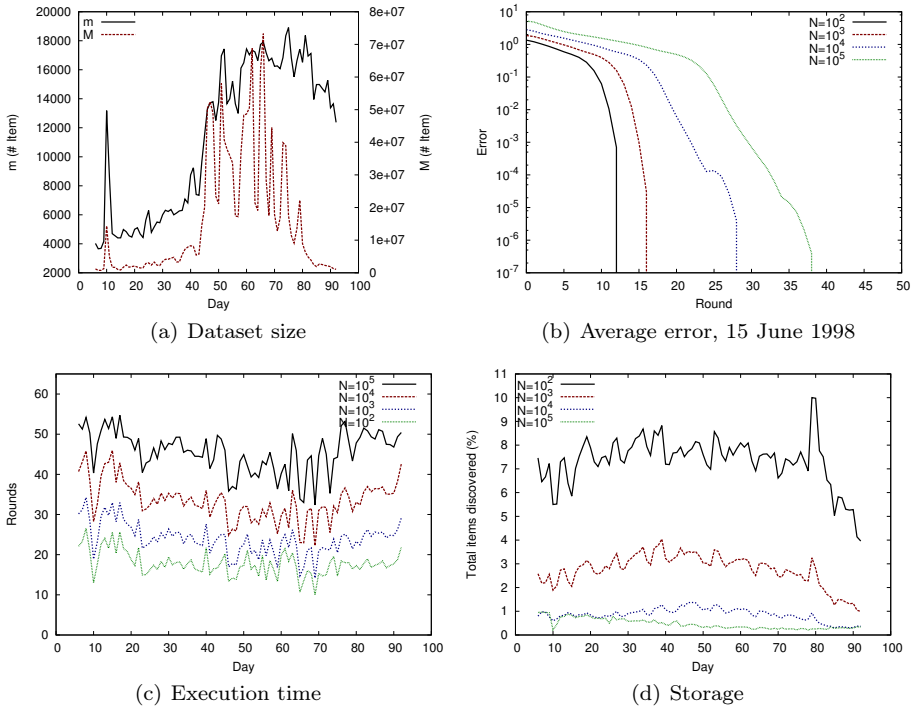
FreqMF grows linearly with d. However, for small values of d, not all nodes obtain a correct top-k set. Figure 3(b) shows the average error, i.e. the number of MF items not included in the output set when the algorithm terminates. For each value of d between 1 and 10 we plot the results from 100 individual experiments shifted randomly along the x-axis. Experiments where the observed error is strictly equal to zero are not plotted due to the logarithmic y-axis. For small values of d, most experiments end before convergence has been reached and hence the recorded error is high. However, starting from $d = 7$, all nodes in all experiments correctly identify all most frequent items. In the remainder of this paper we thus set $d = 8$.

6.3 Realistic dataset

Experiments described so far are based on artificial distributions. In this section we evaluate FreqMF using a realistic dataset which represents a typical application of top-k monitoring [2]. We consider the problem of HTTP request monitoring across a distributed set of mirrored web servers. We initialize our simulations based on an access log from a popular website set up for the 1998 FIFA Soccer World Cup, one of the world's largest sporting events. The website that was accessed over 1.3 billion times between April 30, 1998 and July 26, 1998, which represents an average of over $11,000$ accesses per minute [1]. In our experiments, each web page is an item and the number of page accesses corresponds to the item frequency. We assume the web site is replicated on a number of web servers and we distribute items (page accesses) randomly between nodes (web servers) in our simulated network.

Figure 4(a) shows the characteristics of the dataset from day 6 to the last day 92 (the first 5 days have very few accesses). The solid line represents the number of distinct items (pages) m and the dashed line represents the total number of items (page accesses) M. Note that the units for m are on the left side of the graph while the units for M are on the right side of the graph.

For each day recorded in the trace, we use FreqMF to determine the most frequently accessed web pages (items). Except very few, marginal cases, we obtain fully correct results in all experiments. Figure 4(b) shows the error during a sample execution corresponding to day June 15th 1998, i.e., the day with the largest number of website accesses. In all considered settings, with the number of web servers (nodes)



(a) Dataset size

(b) Average error, 15 June 1998

(c) Execution time

(d) Storage

Fig. 4 Experiments over the World Cup data set.

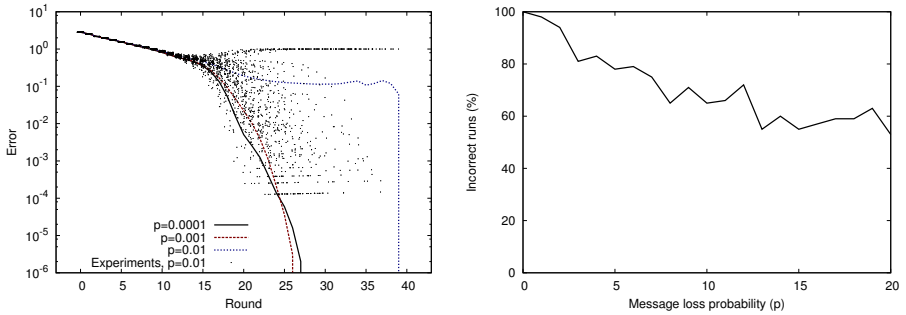
N between 100 and 10,000, the error converges to zero within less than 40 rounds. Figure 4(c) shows the average execution time for the entire data set. Error bars are not shown due to a lack of space, however, the maximum execution time for all the experiments (100 repetitions for each of the 87 days) is 59.

Figure 4(d) shows the storage cost measured as the average percentage of items discovered by a node. Clearly the smaller the network, the more items are known to nodes. Note that the number of distinct items is between 14,624 and 75,632 and the average number of items stored per node is between 945 and 6,968 (for $N = 100$) and between 116 and 248 (for $N = 10,000$).

Communication overhead is not plotted because it is constant throughout all experiments and depends only on s. Each node generates one request per round and receives on average one request per round. Hence, a node sends on average $2 \cdot s$ items per round. To put these numbers into perspective, assuming an ⟨item id, frequency⟩ pair size of 160 bits, round duration of 1 second, and $k = 10$, a node transmits at an average rate of 0.8KB/s. In an experiment running for 50 rounds, a node generates on average 40KB of network traffic.

6.4 Failures

Figure 5(a) shows the behavior of FreqMF in the presence of node failures. At the beginning of the experiment the network consists of 10,000 nodes and at every round



(a) Churn scenario (b) Drop scenario

Fig. 5 Behavior of FreqMF under failure scenarios. $N = 10^4$, World Cup data set

each node fails with probability p, with no substition. For p equal to 0.001 and 0.001 the algorithm correctly identifies all the top-k most frequent items and there is very little impact on convergence speed. Note that probability $p = 0.001$ corresponds to a mean session duration of approximately 15 minutes (assuming a round length of 1 second), which matches the rates observed in real P2P systems [17]. For a churn rate ten times higher, $p = 0.01$, the behavior is different. Convergence is much slower and in 12% of cases only 9 out of 10 frequent items are correctly identified. We show the results from this experiment as an additional scatter plot in Figure 5(a).

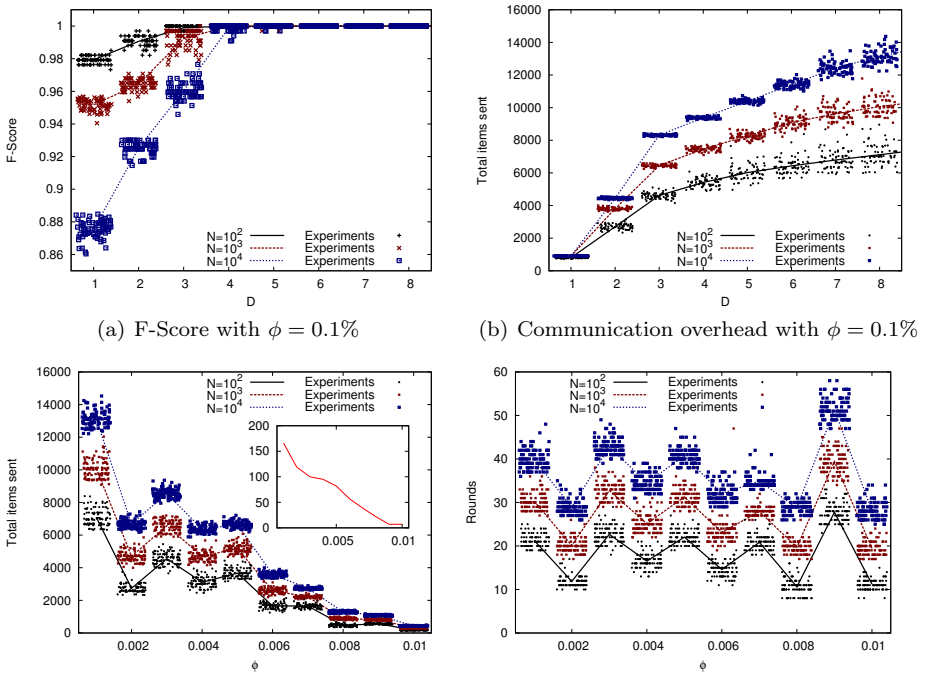
In case of communication failures, the algorithm returns in our settings either a correct item set or misses a single item (i.e., identifies 9 out of 10 items). Figure 5(b) shows the percentage of incorrect runs when messages are lost with probability p between 0% and 20%. Note that such a message loss rate is exceptionally high and real implementation can mitigate the problem by using a reliable transport layer.

6.5 FreqRF

We have focused so far on the FreqMF algorithm, our main contribution in this paper. In this section we briefly evaluate FreqRF. We leave out FreqAF because its behavior is almost identical to FreqRF.

While most of the evaluation metrics such as convergence time and communication overhead are the same as in the previous sections we need to extend the error definition. In FreqMF the size of the target set is constant (k) and so each false negative (missing an item) corresponds to a false positive (selecting an incorrect item). In FreqRF false positives and false negatives are independent. For this reason, we define *precision* as the fraction of the *RF* set correctly identified by the algorithm and *recall* as the fraction of the result set that is actually correct. We evaluate the algorithm using an *F-score* defined as a geometric mean of precision and recall.

Figure 6(a) shows the F-score for FreqRF applied to the World Cup datasets with a relative frequency threshold $\phi = 0.01$ (corresponding to 166 items) and increasing values of the termination threshold d and several network sizes. Individual experiments are marked as dots randomly shifted along the x-axis. The curves indicate average F-scores. Similarly to FreqMF, for d larger than 6 all experiments terminate with no error.



(a) F-Score with $\phi = 0.1\%$

(b) Communication overhead with $\phi = 0.1\%$

(c) Communication overhead with $d = 8$

(d) Convergence time with $d = 8$

Fig. 6 Behavior of FreqRF with different values of d and Φ.

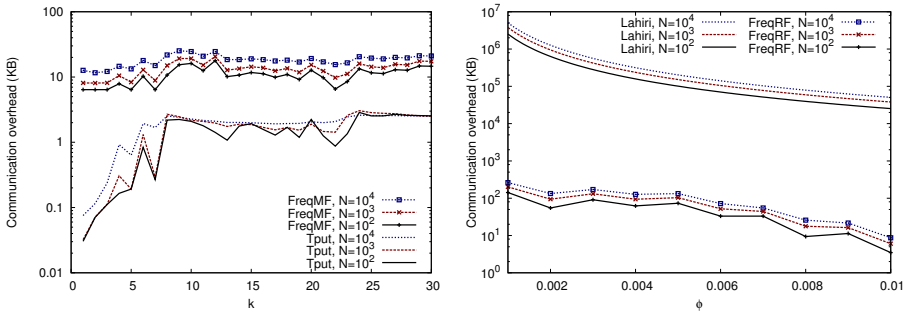
Figure 6(b) shows the communication cost in the same experiments measured as the total number of items exchanged until termination, averaged over all nodes. As expected, the communication cost grows with d. The sweet spot for d appears to be around 8 as in the FreqMF algorithm.

With d fixed at 8, we evaluate the behavior of FreqRF for different values of the relative threshold parameter ϕ. Figures 6(c) and 6(d) show the communication overhead and the convergence time for ϕ between 0.001 and 0.01. Note that Figure 6(c) also shows the correspondence between the threshold ϕ and the size of the RF set. In particular, only 7 items belong to RF when $\phi = 0.01$. Our experiments seem to have a high jitter which is explained by the fact the algorithm must run longer when the ϕ threshold is close to some item frequency $\hat{F}(i)$. In such cases, a higher $\hat{F}(i)$ approximation accuracy is required to determine if item i is above or below the threshold.

7 Related work

Several algorithms have been proposed for finding frequent items in a distributed data set [6], either in the form of an MF set [4,2] or RF and AF sets [12,13,14]. To the best of our knowledge, this paper is the first one to describe a generic scheme that addresses all three variants of the problem.

The existing approaches can be divided into three classes based on the communication architecture: master-slave [4,2], hierarchical [13,14], and fully decentralized [12].



(a) *MF* problem (b) *RF* problem

Fig. 7 Communication overhead, per node, of FreqMF and FreqRF compared against state-of-the-art algorithms.

The master-slave algorithms have a common drawback that the master node is a single point of failure and potentially a performance bottleneck. For instance, TPUT is master-slave protocol proposed by Cao and Wang [4] which consists of three request-response phases. In the first phase, master asks all slave nodes hosting the distributed data set for their current top-k items and computes a lower bound on the local frequency for the k-th item. In the second phase, master asks the slaves for the frequencies of items above the lower-bound and refines the candidate top-k set. In the last phase, master obtains from the slaves the final *MF* set.

Figure 7(a) compares the communication overhead, per node, generated by the execution of TPUT and FreqMF on a one-day subset (June 15th, 1998) from the World Cup dataset. TPUT generates one order of magnitude less traffic than FreqMF, however, it is important to note that it sends all messages to a single node – the master. In a network of $10,000$ nodes the master would for example receive more than 25MB of traffic. In contrast, our approach is fully symmetric and every node receives on average the same load.

The netFilter protocol [13] and the algorithm described by Manjhi et al. [14] are examples of hierarchical approaches. The netFilter divides items into groups and uses a hierarchical aggregation algorithm to select frequent item candidates. It then runs a second-stage algorithm to select a final result set amongst the candidate items. Manjhi et al. [14] aggregate (ϵ, α)-synopsis data structures over a node hierarchy and adjust the data compression rate using a precision gradient to minimize the total communication cost. Both these approaches require a separate control plane to maintain a node hierarchy. Furthermore, both approaches are prone to failures of nodes and links close to the root in the hierarchy. In contrast, our algorithm is fully decentralized, has no performance of reliability bottlenecks because all nodes share exactly the same responsibilities, and requires a very simple membership model which allows it to scale to extremely large systems.

Lahiri and Tirthapura [12] proposed two gossip protocols based on random sampling that solve the *RF* and *AF* problems in a fully decentralized way. Their approach is based on the observation that a frequent item is likely to occur at an approximately the same relative frequency in an appropriately sized random sample. To give guaranteed accuracy, a large enough sample has to be inspected. The sample size is not known in advance and thus the protocols adaptively select a sampling probability based on

min-wise independent permutations [3]. The algorithm has two parameters: the approximation error ψ and the error probability δ. An item i is considered frequent if $\hat{F}(i) \geq \phi$ and infrequent if $\hat{F}(i) < \phi - \psi$. When the algorithms is executed long enough, the following probabilistic guarantee holds: with probability at least $1 - \delta$ every node reports all frequent items and does not report any infrequent items. The algorithm runs for $4N \log N$ rounds and each node exchanges $t = \frac{0.25}{\psi^2} \ln \frac{2}{\delta}$ items per round. Thus for small ψ and δ the algorithm incurs a large communication cost. Figure 7(b) shows the communication cost for FreqRF and Lahiri and Tirthapura's algorithms applied to a one-day subset (again, June 15th 1998) of the World Cup dataset, where we conservatively assume a low result accuracy ($\delta = 0.1$ and $\psi = \phi/2$). Clearly, FreqRF has a multiple orders of magnitude lower cost.

Finally, we note that some algorithms address a streaming variant of the frequent item discovery problem in which item frequency is measured in a data stream using a sliding window. In particular, Babcock and Olston [2] address the streaming problem only and Manjhi et al. [14] tackle both the streaming and static-set problems. We are planing to adapt our algorithm to the streaming model in our future work.

8 Conclusions

FreqMF, FreqAF and FreqRF are a family of gossip algorithms that efficiently identify the most frequently occurring items in a massively distributed data set. Unlike current state-of-the-art approaches, these algorithms are fully decentralized, have no performance or reliability bottlenecks, and require a very simple membership model, which together allow it to scale to extremely large and dynamic systems. Using a combination of analytical models and experimentation with real-world data traces, we show that our algorithms are fast, highly scalable, and resilient to churn.

References

1. M. Arlitt and T. Jin. 1998 World Cup web site access logs, Aug. 1998. Available at `http://www.acm.org/sigcomm/ITA/`.
2. B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of the Int. Conf. on Management of data (SIGMOD'03)*, pages 28–39, San Diego, CA, 2003. ACM.
3. A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *Proc. of the 38th ACM Symposium on Theory of computing (STOC'98)*, pages 327–336, Dallas, Texas, 1998. ACM.
4. P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proc. of the 23rd Symposium on Principles of distributed computing (PODC'04)*, pages 206–215, St. John's, Newfoundland, Canada, 2004. ACM.
5. M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.
6. G. Cormode. Continuous distributed monitoring: a short survey. In *Proc. of the 1st Int. Workshop on Algorithms and Models for Distributed Event Processing (AlMoDEP'11)*, pages 1–10, Rome, Italy, 2011. ACM.
7. P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In J. M. Abello and J. S. Vitter, editors, *External memory algorithms*, pages 39–70. American Mathematical Society, Boston, MA, USA, 1999.
8. L. Golab, D. DeHaan, E. D. Demaine, A. Lopez-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement (IMC'03)*, pages 173–178, Miami Beach, FL, USA, 2003. ACM.

9. M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM TOCS*, 23(3):219–252, 2005.
10. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM TOCS*, 25(3), Aug. 2007.
11. C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou. Dynamically maintaining frequent items over a data stream. In *Proceedings of the 12th international Conference on Information and Knowledge Management (CIKM'03)*, pages 287–294, New Orleans, LA, USA, 2003. ACM.
12. B. Lahiri and S. Tirthapura. Identifying frequent items in a network using gossip. *J. Parallel Distrib. Computing*, 70(12):1241–1253, Dec. 2010.
13. M. Li and W.-C. Lee. Identifying frequent items in P2P systems. In *Proc. of the 28th Int. Conf. on Distributed Computing Systems (ICDCS'08)*, pages 36–44. IEEE, 2008.
14. A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proc. of the 21st International Conference on Data Engineering (ICDE'05)*, pages 767–778. IEEE, 2005.
15. A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, Sept. 2009.
16. M. Raab and A. Steger. "Balls into bins" — a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, volume 1518 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 1998.
17. D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, pages 189–202, Rio de Janeriro, Brazil, 2006. ACM.