

ProFID: Practical Frequent Item Set Discovery in Peer-to-Peer Networks

Emrah Çem¹, Öznur Özkasap^{1*}

Department of Computer Engineering
Koç University, Istanbul, Turkey
{ecem|oozkasap}@ku.edu.tr

Abstract. This study addresses the problem of discovering frequent items in unstructured P2P networks. This problem is relevant for several distributed services such as cache management, data replication, sensor networks and security. We make three contributions to the current state of the art. First, we propose a fully distributed Protocol for Frequent Item Set Discovery (ProFID) where the result is produced at every peer. ProFID uses a novel pairwise averaging function and network size estimation together to discover frequent items in an unstructured P2P network. We also propose a practical rule for convergence of the algorithm. Finally, we evaluate the efficiency of our approach through an extensive simulation study on PeerSim and present our conclusions.

Keywords: peer-to-peer, distributed computing, gossip, frequent items.

1 Introduction

1.1 Motivation

Recent years have witnessed an extraordinary growth of P2P network services that have a very dynamic structure since peers may join or leave the system at any time. On the other hand, advantages of these type of systems are the enhanced scalability and service robustness due to their distributed architectures. Because of P2P systems' dynamic and scalable nature, centralized approaches are not as functional and reliable as decentralized approaches. In decentralized approaches, there is no central administration, so peers need to communicate with each other to perform various tasks such as searching. Furthermore, peers may need a system-wide information such as network size, system load, query/event counts, or mostly contacted peers for specific files in order to perform various tasks such as load-balancing or topology optimization [1]. Database applications, wireless sensor networks, and security applications can also make use of frequent item set discovery, as well as P2P applications. Hence, efficient discovery of frequent items would be a valuable service for distributed systems.

* Research supported by TUBITAK (The Scientific and Technical Research Council of Turkey) under CAREER Award Grant 104E064.

There are various P2P applications such as cache management, search technique design, query refinement, content mirroring, network topology optimization, denial of service attack and internet worm detections that can utilize frequent item set discovery service. For example, cache management can be adapted to a frequent (popular) item set discovery problem. Since the probability of accessing frequent items in the future is more probable than accessing non-frequent items, caching frequent items will reduce the average access time significantly. Other than p2p networks, frequent item set discovery protocol can also be used in sensor networks to detect anomalies and attacks. For example, let's assume we have many movement sensors implanted in the ground which can detect whether there is a moving object around it or not. If most of the sensors detect a moving object, then there might be the possibility of an attack. It can also be used to detect anomalies in atmospheric conditions via temperature sensors [2, 3]. Moreover, that problem can be used in computing and answering iceberg queries as well as identifying large network flows [4]. Frequent item set discovery problem is also applicable for database applications [5, 6].

1.2 Related Work

Frequent item set discovery problem has been studied for data streams, P2P services and network monitoring. First deterministic algorithm for data streams is proposed in [5]. Algorithms named *Sticky sampling* and *Lossy counting* for identifying items whose frequency exceeds a user-defined threshold value are proposed in [7]. They find approximate frequencies, and the error rate is bounded by a user-specified parameter. They also propose an algorithm to optimize finding frequent item set in a single pass. However, if the dataset is highly skewed, algorithm may result in a high error rates. Even though their algorithms are efficient and applicable for the discovery of frequent items in datastreams, it is a centralized algorithm. More related to our work, [4] proposes an algorithm to find approximate frequent items in distributed data streams. Datastreams are composed of item occurrences with time stamps and recent items are given more importance while discovering frequent items. The aim is to output approximate frequencies of items whose true frequencies exceed a user-specified threshold. In order to accomplish that, a hierarchical communication structure is constructed. Moreover, the concept of precision gradient is introduced in order to minimize communication overhead. Another recent work [8] describes significant algorithms in frequent item set discovery in data streams and provides baseline implementations of them. Experimental evaluation on different data sets is also performed in order to figure out how practical their implementations are. An algorithm for the problem of distributed monitoring of thresholded counts is proposed in [3, 9]. The goal is to monitor all items whose frequency is above a threshold value within specified accuracy bounds. They use hierarchical approach. However, since the frequent items are only available on a single node, this work is exposed to single point of failure.

Obtaining exact frequent items with a technique named in-network filtering is investigated in [1]. It consists of two phases called candidate filtering and

candidate verification. In order to calculate the aggregates, a tree-based hierarchy composed of only stable peers is formed, and the most stable peer is chosen as the root of the hierarchy. This is done to make algorithm more stable, since constructing tree-like structures is not robust against failures. The basic idea is to form item groups and calculate the aggregates of item groups. If item group frequency is below the threshold value, than all items in that group fails to be a candidate, otherwise items are chosen as candidates (for being frequent item). They theoretically show how to optimize parameters of the proposed approach such as group size, filter size, and give the effects of parameters on the performance of the algorithm. However, this approach is not practical and easy to deploy for large scale distributed settings.

A uniform gossip based technique for disseminating the local information of each peer is proposed in [10]. Since they use gossiping for data dissemination, result is probabilistic as opposed to the study of [1]. However, thanks to gossiping, there is no underlying network structure or central control which makes algorithm simpler. In order to identify frequent elements, a thresholding mechanism is used, and by using sampling, the communication load is decreased. Moreover, a space efficient data representation named *sketch* is used to store aggregate values efficiently.

There are also various studies focusing on computing aggregates in distributed systems. Jelasity et al. [2] present a fully distributed way of calculating aggregates such as counting, averages, sums, products, and extremal values. They restart the gossip protocol periodically in order to prevent accumulation in estimation error. They also give both theoretical analysis and empirical results of convergence of the proposed algorithm. Moreover, they analyze the effect of overlay topology on convergence of the algorithm, and show that it effects the convergence significantly. Hence, they use topology information while determining the termination time. This is not practical since it may not be available at all peers. Kempe et al. [11] propose a push-synopses protocol for aggregate computation and analyzes the scalability, reliability and efficiency of their approach. Algorithm converges to true average, only if all peers have a knowledge about all items in the system, which might be an inconvenient requirement for large networks without centralized agents. Moreover, efficiency of their algorithm uses the assumption of uniform gossiping. [12] also uses similar approach except that they assume wireless broadcast is available. Chitnis et al. [13] introduces a hybrid solution to compute aggregates in sensor networks, which uses strengths of both gossip and tree aggregation protocols together.

1.3 Our Contributions

We address the problem of frequent item set discovery in unstructured P2P networks. Our contributions are as follows:

1. We propose a fully distributed gossip-based approach named ProFID using pairwise averaging function which is novel in frequent item set discovery problem.

2. We introduce a practical convergence algorithm. In contrast to previous works, each peer gives local decision for convergence based on the change of updated local state. Converged peers may list frequent items independently from other peers in the network.
3. We developed a model of ProFID on PeerSim [14], and performed various experiments to compare and measure its efficiency and scalability.

Outline. This paper is organized as follows. In the next section, we describe the system model and define the problem. In Section III, we give the principles and algorithms of ProFID. In Section IV, we provide results from experimental evaluation. Finally, we conclude the paper and state the future directions in Section V.

2 System Model and Problem Statement

2.1 System Model

We consider a network consisting of N peers which only know their own states (local view) initially. In order to have knowledge about all peers' states (global view), peers have to collaborate and obtain information about other peers' states in the network. We assume that peers form an unstructured P2P network. In order to collaborate and communicate, peers need to know the identifiers of their neighbors. This neighborhood relation determines the topology of an *overlay network*. We also assume that peers may leave (intentionally or due to failures) or join the network at any time, which is inevitable in P2P networks.

In distributed computing, communication is determined by a time model and it can be categorized as synchronous and asynchronous. In synchronous time model, local clocks of peers are synchronized and each peer performs operations within the same time interval. This is actually not practical in large distributed systems due to the variable and unpredictable message delays. In asynchronous time model, each peer uses its local clock to perform a computation. In our case, peers communicate in rounds with a fixed duration. However, it is not necessary to synchronize the peers' local clocks because peers use clocks just to perform periodic operations. Round duration just determines how often a peer sends local state to its neighbor(s).

2.2 Problem Statement

There are many problems that may need information about globally frequent items in a distributed system. Hence, efficiently computing frequent items in distributed systems is valuable. DDoS attack detection, internet worm detection, cache management, search technique design, P2P media streaming and replication protocols [1] are some example applications.

Consider a network consisting of N peers denoted as $P=\{P_1, P_2, \dots, P_N\}$ and M item types denoted as $D=\{D_1, D_2, \dots, D_j, \dots, D_M\}$, where D_j has a global

frequency g_{D_j} . Parameters N , M , and g are system-wide information, hence they are unknown to all peers a priori.

Let each peer (P_i) has a local set of items $S_i \subseteq D$ and each local item (D_j) has a local frequency f_{i,D_j} such that

$$g_{D_j} = \sum_{i=1}^N f_{i,D_j}, \quad D_j \notin S_i \implies f_{i,D_j} = 0 \quad (1)$$

To illuminate the parameters, let's examine how we can use frequent item set discovery protocol in a cache management problem. Consider three peers in Fig.1 representing local movie databases distributed in different countries. Items at each peer correspond to the name of movies queried/searched by clients and frequency of each item corresponds to the number of queries that includes the item (movie name in this case). Our system parameters for these peers can be written as given in Table 1. After running a frequent item set discovery protocol with threshold 8000, all the movies that are queried more than the threshold are computed. In this example, protocol needs to output *Avatar* and *The Hurt Locker*. Peers may actually utilize this information to cache frequently queried/searched movies in order to access them in a shorter time because clients will probably query these movies more than others in future, at least for a period of time.

Item	Frequency	Item	Frequency	Item	Frequency
Avatar	: 4500	The Hurt Locker	: 6000	Avatar	: 500
The Hurt Locker	: 3200	Avatar	: 5000	The Hurt Locker	: 200
Disaster Movie	: 100	Epic Movie	: 50	Disaster Movie	: 600
Epic Movie	: 20				
P₁		P₂		P₃	

Fig. 1: Sample three peers with local frequencies of movie items queried/searched

3 ProFID: Protocol for Frequent Item Set Discovery

We provide a gossip-based fully distributed approach with pairwise averaging function for discovering frequent items in unstructured P2P networks. Utilizing pairwise averaging function with gossip-based aggregation and a practical convergence rule is novel and beneficial. Due to the unstructured form of communication in gossiping, it is not easy to prevent a local frequency of an item at a peer from being accounted for multiple times. For this reason, either peers need to handle the duplicates in incoming gossip messages or use an aggregation function which is insensitive to duplicates. Pairwise averaging function is

Global parameters (unknown a priori):
$P = \{P_1, P_2, P_3\}$, $N=3$
$D = \{Avatar, DisasterMovie, TheHurtLocker, EpicMovie\}$, $M=4$
$g(Avatar) = 11000, g(DisasterMovie) = 700, g(TheHurtLocker) = 8400, g(EpicMovie) = 70$
P₁'s local parameters:
$S_1 = \{Avatar, TheHurtLocker, DisasterMovie, EpicMovie\}$
$f_{1,Avatar} = 4500, f_{1,TheHurtLocker} = 3200, f_{1,DisasterMovie} = 100, f_{1,EpicMovie} = 20$
P₂'s local parameters:
$S_2 = \{TheHurtLocker, Avatar, EpicMovie\}$
$f_{2,TheHurtLocker} = 3000, f_{2,Avatar} = 2500, f_{2,EpicMovie} = 50$
P₃'s local parameters:
$S_3 = \{Avatar, TheHurtLocker, DisasterMovie\}$
$f_{3,Avatar} = 500, f_{3,TheHurtLocker} = 200, f_{3,DisasterMovie} = 600$

Table 1: Parameters of distributed movie database example

the latter choice. Even though the local frequency of an item at a peer might be accounted for multiple times, it still efficiently converges to the approximate global average frequency of the item at each peer. The only assumption we do is to perform pairwise averaging operation atomically. We used buffering and timeout mechanisms to perform pairwise averaging atomically.

Being a frequent item is directly related with the global frequency of that item. Likewise, it is also directly related with the global average frequency of the item, where global average frequency of an item is defined as:

$$ga(D_j) = \frac{g(D_j)}{N} = \frac{1}{N} \sum_{i=1}^N f_{i,D_j} \quad (2)$$

A new threshold value which will be compared by the global average of the item (instead of item's global frequency) while determining frequent items can be defined as:

$$\Delta = \frac{T}{N} \quad (3)$$

Using this parameter, we may rewrite the frequent item set as

$$F(\Delta) = \{D_j | ga(D_j) > \Delta, \forall j \in 1, 2, \dots, M\} \quad (4)$$

System size N is a global parameter, so it is not known by any peer a priori and it also needs to be calculated. In order to calculate system size, an initiator peer adds a unique item type named ui in its local item set. The local frequency of this item is set to 1. Since only one peer has that unique item, average frequency of that item would converge to $\frac{1}{N}$ from which Δ can be calculated simply by multiplying the converged result with the threshold value. [2]

In general, gossip algorithms can be divided into 3 parts in terms of decisions regarding:

1. To whom gossip messages to send
2. What to perform when a message comes in
3. When to stop (convergence rule)

In ProFID, all of those decisions are taken locally, and peers do not know any system wide information such as topology and network size, initially. Algorithm 1 shows initialization, periodic send operation and handling of incoming messages. Convergence rule is explained in Sect. 3.2 in detail. Those algorithms are the general operations, details such as buffering and timeout mechanisms were excluded for better readability. Description of algorithm parameters are given in Table 2.

Algorithm 1: *ProFID: Protocol for Frequent Item Set Discovery*

Input: $fanout, mms, ui, convLimit, \varepsilon, T, S$
Output: F : set of frequent items

Initialize
if *Initiator* **then**
 $S.add(ui,1);$
 $converged=false; prevSizeEstim=0; convCounter=0;$
 1.Gossip(periodically do)
 if $!converged$ **then**
 $targets = getNeighbors(fanout);$
 for $i=1:fanout$ **do**
 $send(push, message(S,mms), targets(i));$
 2.Handle incoming messages
 $messg=accept();$
 if $messg.Type == push$ **then**
 $avg = AVERAGE(S, messg); S.update(avg); send(pull, avg, sender)$
 else if $messg.Type == pull$ **then**
 $S.update(messg);$
 $currSizeEstim=messg.getVal(ui);$
 if $ISCONVERGED(convLimit, \varepsilon)$ **then**
 $converged=true;$
Query
 $\Delta = T * currSizeEstim; F = \{item \mid \forall item \in S, S.getFrequency(item) > \Delta \};$

Definition 3.1 *Epsilon condition:* A peer satisfies this condition if the frequency of ui at that peer changes at most ε percentage after a gossip.

3.1 Atomic Pairwise Averaging

In order to calculate global frequencies of items, we use pairwise averaging function with network size estimation. Our pairwise averaging function uses push-pull

Parameter Type	Parameter Name	Description
User defined	T	threshold
Protocol	$fanout$	number of peers to whom gossip message is sent at each round
Protocol	mms	maximum gossip message size a peer can send.
Protocol	ui	unique item used in network size estimation
Convergence	$convLimit$	number of successive rounds a peer needs to satisfy epsilon condition (See Def. 3.1) in order to converge
Convergence	$\varepsilon(epsilon)$	parameter used to determine epsilon condition

Table 2: Algorithm parameters

scheme meaning that a peer sends its state (in a push message) to a target peer and the target peer performs averaging operation using its own state and incoming state, then replies the average of incoming items (in a pull message) back to the sender. Then, sender updates its state. By this way, a single push-pull based pairwise averaging operation is completed. In order to prevent misleading calculations this operation must be performed atomically. Fig. 2a illustrates the order of an example of push-pull based non-atomic pairwise averaging operation. Assume that system has a unique item named $item1$ with global frequency 10 and initial local frequencies are $f_{1,item1}=3$, $f_{2,item1}=5$, $f_{3,item1}=2$. The updated local frequencies of $item1$ after each operation shown in Fig. 2b. When states of all peers are considered after operation 4, Eq. 1 does not hold any more, which puts the system in an inconsistent state. Thus, even though no addition/removal of item, or loss of a message occurs, global value of item changes.

Since the latencies of messages are very dynamic in real life scenarios, we can not control the message order without a buffer mechanism. We used a buffer mechanism in such a way that whenever a peer sends a push gossip message to another peer, all the requests coming into that peer or push operations to other neighbors (in case of $fanout > 1$) are buffered until the reply of the push message comes in. To deal with message losses, we use a timeout mechanism. For this example, P_1 sends push message to P_2 and starts waiting for the corresponding pull message. Push message came from P_3 is not replied immediately, but buffered. Whenever the reply comes into P_1 from P_2 , P_1 updates its state, then removes and performs the next event in the buffer which is sending pull message to P_3 . After sending the pull message, P_3 receives the pull message and updates its state. By means of buffering and timeout mechanisms, we performed push-pull based pairwise averaging operation atomically (consecutively).

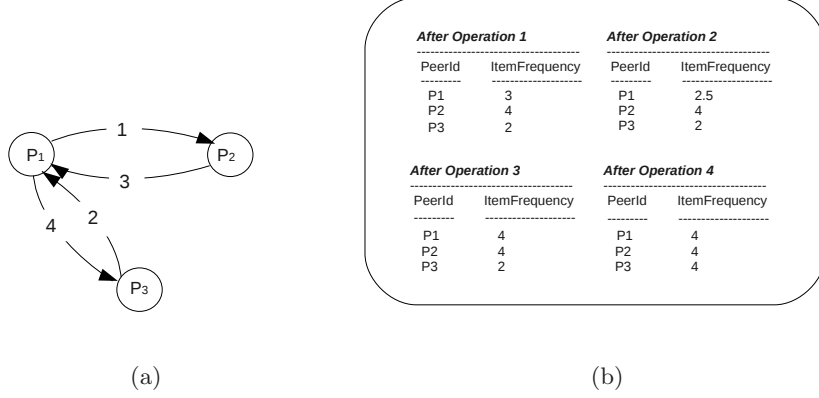


Fig. 2: (a) Illustration of operation order of a non-atomic pairwise averaging
(b) States of peers after each operation

3.2 Convergence Rule

In ProFID, peers use parameters ϵ and *convLimit* to determine whether algorithm converged and results are available. The idea of convergence is simply to calculate *similar frequency* values within at least a time length of *convLimit* gossip rounds. Here calculating *similar frequency* means getting two average frequency estimation values that change at most ϵ percentage in consecutive rounds of algorithm. When a *similar frequency* is calculated, then a counter, starting from zero, is incremented. Otherwise, counter is reset to zero. Whenever, the counter reaches *convLimit* value, then peer decides that algorithm converged and can use frequent item set in a required service. As noted in [2], initial distribution of an item does not affect the convergence speed, hence we use *ui* value's average frequency estimation to determine the convergence of the algorithm.

Convergence Rule. Let S_t be the average frequency estimation of item *ui* at time t . Following similarity check is performed at each gossip round and *convCounter*, starting from zero, is set accordingly. Whenever the *convCounter* reaches to *convLimit*, then peer decides that algorithm has converged.

$$\begin{aligned}
\text{convCounter} &= \text{convCounter} + 1 && \text{if} && 100 \left| \frac{S_t - S_{t-1}}{S_{t-1}} \right| \leq \epsilon \\
\text{convCounter} &= 0 && \text{otherwise} &&
\end{aligned}$$

In order to prevent misleading convergence cases at the beginning of the algorithm due to the similar initial states of neighbors, we keep *convLimit* not

Algorithm 2: *ISCONVERGED: the convergence rule for ProFID.*

Input: *convLimit*, ε **Output:** *converged*:boolean representing convergence

```
if  $\varepsilon \geq (\text{currSizeEstim} - \text{prevSizeEstim})/\text{prevSizeEstim} \ \&\& \ \text{currSizeEstim} \neq 0$  then
   $\perp$  convCounter++
else
   $\perp$  convCounter = 0
  prevSizeEstim = currSizeEstim;
return converged || convCounter==convLimit
```

less than 10 so that peers can not decide to converge in the early stages of the algorithm.

Since we design a fully distributed protocol, we make a local decision about convergence of algorithm. In terms of state exchanges, our gossip operations for averaging are similar to [2] and the convergence issue has been analyzed clearly in that study. However, the proposed termination criteria uses topology information which may not be available at peers initially. On the other hand, ProFID uses no system-wide information which makes it more practical.

As another related work, [11] gives a probabilistic upper bound on the number rounds for all peers to converge. This upper bound depends on network size and accuracy. It is shown theoretically that the more the algorithm runs, the more probable to get correct estimations. However, the problem in the proposed algorithm is that peers may not have knowledge about all items in the network, which will cause accuracy drop in the algorithm.

4 Simulation Results

We used PeerSim [14] simulator to build the model for ProFID. It is a modular and very scalable simulator and gives us a way to access and configure transport layer properties such as message loss and delays. We evaluated the behavior and performance of ProFID through extensive large-scale distributed scenarios (up to 30,000 peers) on PeerSim. Random graphs with average degree 10 is used in the experiments. Moreover, all the simulation data points presented in graphs are the average of 50 experiments. We consider the following performance metrics in our analysis:

- **epsilon and convLimit:** The effects of convergence parameters have been analyzed in terms of number of rounds to converge and average number of messages sent per peer. Relative error has also been analyzed for different combination of convergence parameters.
- **number of rounds (to converge):** This performance metric measures how fast the algorithm converges. The effects of convergence parameters,

average degree of nodes, and number of nodes on convergence speed has been analyzed

- **messages sent per peer:** This performance metric measures the energy efficiency of the algorithm, the effects of convergence parameters and average degree of nodes on energy efficiency has been analyzed
- **percentage of converged peers:** This metric is another measure of how fast algorithm converges. It shows the percentage of peers that converged during gossip rounds. The effects of this metric has been analyzed in terms of convergence parameters and network size
- **C:** This parameter is the convergence time constant. The effects of this metric has been analyzed in terms of network size and percentage of peer that converged.
- **fanout:** This parameter defines the number of peers to whom gossip message will be sent at each round, its effects on convergence speed and C value has been analyzed.

4.1 Efficiency of Pairwise Averaging

We evaluate the performance of pairwise averaging function by excluding the convergence rule of ProFID. A peer compares the actual averages of items with peers' estimated averages while deciding convergence. This is for better evaluation of the performance of pairwise averaging.

Fig. 3a depicts the scalability of ProFID in terms of time complexity. Number of gossip rounds needed for the convergence of all peers in the system is measured to examine the time complexity of ProFID as the network size scales up. These results confirm the scalability of our system in terms of time needed for convergence. In fact, our results (for number of rounds to converge) agree with the $O(\log N)$ time complexity of epidemic dissemination.

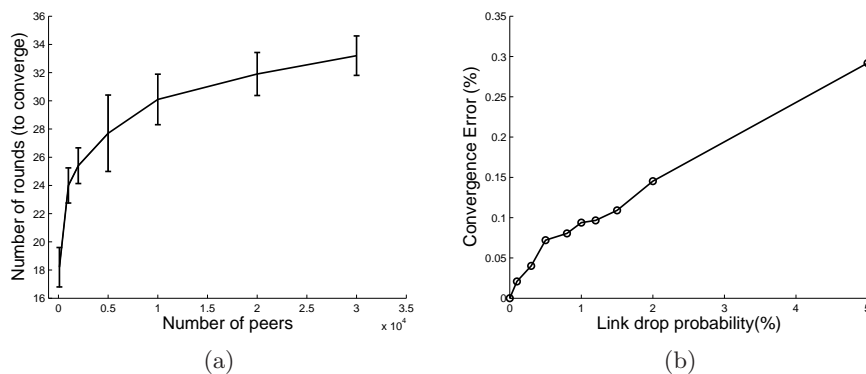


Fig. 3: (a) Number of gossip rounds needed for all peers to converge. (b) The effect of link drop probability on the accuracy of pairwise averaging

Fig. 3b illustrates the convergence error of the pairwise averaging in case of message losses. In this simulation, the message loss probability of each link is independent and identically distributed. Relative error is even less than %1, in case of %5 message drop probability, which show that pairwise averaging is robust against message losses.

Fig. 4a illustrates the effect of C , convergence time constant, on the network size. It decreases with the increasing network size from which we can conclude that for large networks, $\log N$ value will be much more dominant than C . Fig. 4b shows the effect of C on the number of peers that converges. It is actually another way of showing that C value decreases with increasing network size. Thus, larger networks reaches to full convergence for smaller values of C .

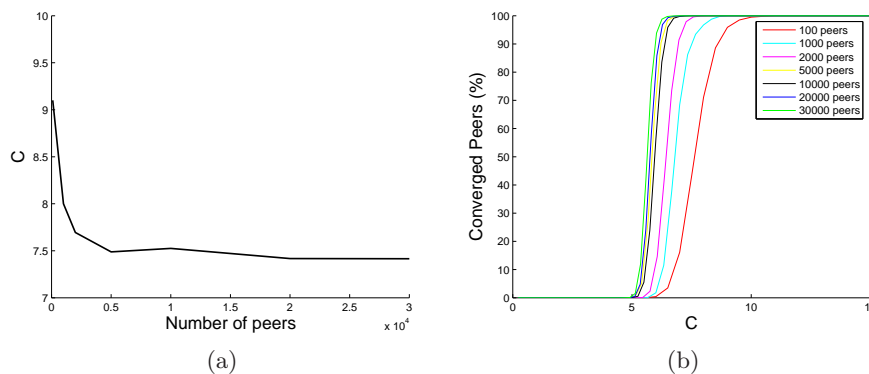


Fig. 4: (a)The minimum value of C such that after $C \log N$ rounds all peers converge. (b)Percentage of peers converged after $C \log N$ rounds

The effect of fanout on convergence speed is illustrated in Fig. 5a. For larger values of fanout, algorithm converges faster since a peer exchanges its state with more neighbors and its state is disseminated faster to the network. The effect of fanout on C value is depicted in Fig. 5b. For larger values of fanout, C value decreases due to the faster convergence of algorithm.

4.2 Efficiency of ProFID

We evaluate the effects of convergence parameters (ε and $convLimit$) and average degree on the efficiency of ProFID. Fig. 6a shows that increasing ε , decreases both average number of messages sent per peer and number of rounds to converge because convergence rule increments $convCounter$ value with more probability, which results in faster convergence. Since algorithm converges faster, peers communicate less and average number of messages sent per peer decreases. In contrast to ε parameter, increasing $convLimit$ increases both the average number of

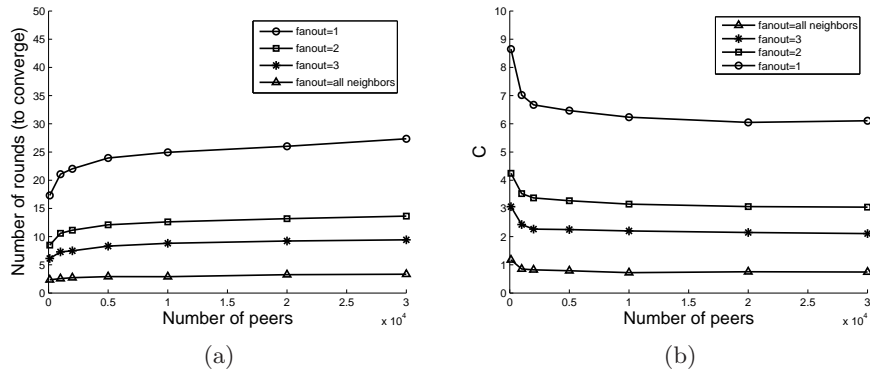


Fig. 5: (a)The effects of fanout on the number of rounds needed for all peers to converge (b)The effects of fanout on C value

messages sent per peer and number of rounds to converge because $convCounter$ needs to be incremented more to reach $convLimit$.

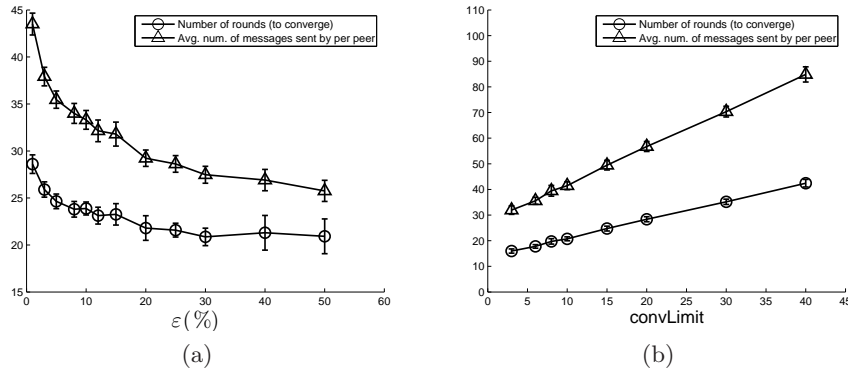


Fig. 6: The effects of ϵ (a) and $convLimit$ (b) on the number of rounds to converge and average number of messages sent per peer

Fig. 7a illustrates the effects of convergence parameters, ϵ and $convLimit$ on the number of rounds to converge. The fastest convergence occurs whenever ϵ parameter takes its largest value and $convLimit$ takes its smallest value, which agrees with the convergence rule. However, there is a tradeoff between convergence speed and accuracy. Finding the optimum values of convergence parameters is an ongoing work. Fig. 7b illustrates the effect of average degrees of peers on the number of rounds to converge. Increased connectivity of the network,

decreases the number of rounds to converge. This actually shows that, optimum results would be taken in complete graphs, as Kempe et al. [11] showed.

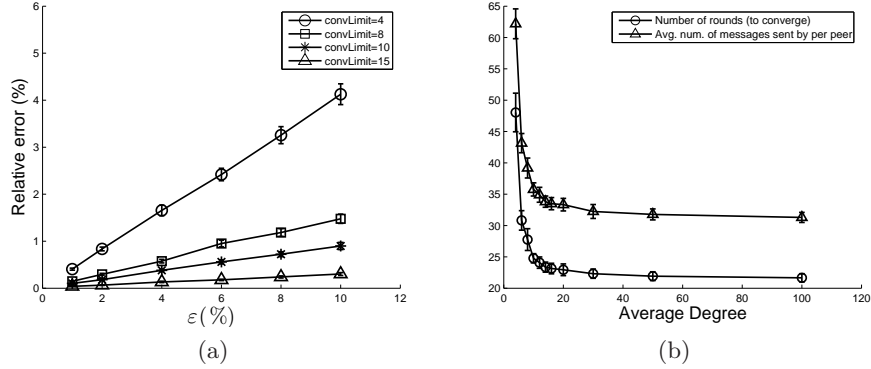


Fig. 7: (a) The effects of the convergence parameters on number of rounds to converge. (b) the effects of average degree on number of rounds to converge and average number of messages sent per peer.

5 Conclusion and Future Work

This paper has proposed a practical protocol named ProFID for discovering frequent items in P2P networks. In contrast to prior work, ProFID uses atomic pairwise averaging function with gossip-based aggregation. It is fully distributed, uses local peer information plus neighborhood knowledge only, and also offers a convergence rule for efficiently approximating system wide averages of data items. We have developed the simulation model, evaluated the behavior and performance of ProFID through extensive large-scale distributed scenarios (up to 30,000 peers) on PeerSim. First, we have studied the atomic pairwise function in terms of its efficiency in approximating averages of items. Then, we have evaluated the protocol by considering several metrics such as epsilon, convergence limit, fanout, number of rounds, and messages sent per peer. The results confirm the practical nature, ease of deployment and efficiency our approach. As future directions, we aim to evaluate ProFID in peer churn scenarios, and investigate the effect of limited gossip message sizes. For comparison, we are developing the well-known push-synopses protocol [11] by adapting it to the problem of frequent item set discovery and practical P2P network settings. Furthermore, we aim to conduct network tests of ProFID on the PlanetLab.

References

1. M. Li and W. C. Lee, "Identifying frequent items in p2p systems," in *Distributed Computing Systems, 2008. ICDCS '08. The 28th International Conference on*, June 2008, pp. 36–44.
2. M. Jelasity and A. Montresor, "Epidemic-style proactive aggregation in large overlay networks," in *ICDCS*, 2004, pp. 102–109.
3. R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *SIGMOD Conference*, Jun. 2006, pp. 289–300.
4. A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston, "Finding (recently) frequent items in distributed data streams," in *Proc. of International Conference on Data Engineering (ICDE)*, Apr. 2005, pp. 767–778.
5. J. Misra and D. Gries, "Finding repeated elements," *Sci. Comput. Program.*, vol. 2, no. 2, pp. 143–152, 1982.
6. R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, pp. 51–55, 2003.
7. G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB*, 2002, pp. 346–357.
8. G. Cormode and M. Hadjieleftheriou, "Finding the frequent items in streams of data," *Commun. ACM*, vol. 52, no. 10, pp. 97–105, 2009.
9. C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *SIGMOD Conference*, 2003, pp. 563–574.
10. B. Lahiri and S. Tirthapura, "Computing frequent elements using gossip," in *SIROCCO*, 2008, pp. 119–130.
11. D. Kempe, A. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," in *Proc. of Symposium on Foundation of Computer Science (FOCS)*, Oct. 2003, pp. 482–491.
12. J.-Y. Chen, G. Pandurangan, and D. Xu, "Robust computation of aggregates in wireless sensor networks: Distributed randomized algorithms and analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 987–1000, 2006.
13. L. Chitnis, A. Dobra, and S. Ranka, "Aggregation methods for large-scale sensor networks," *ACM Trans. Sen. Netw.*, vol. 4, no. 2, pp. 1–36, 2008.
14. "The Peersim simulator," <http://peersim.sf.net>.