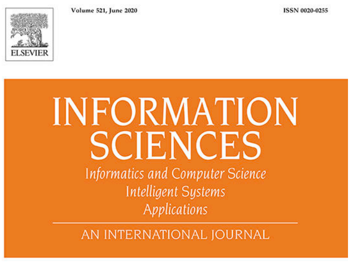


Distributed mining of time–faded heavy hitters

Marco Pulimeno, Italo Epicoco, Massimo Cafaro

PII: S0020-0255(20)30959-2
DOI: https://doi.org/10.1016/j.ins.2020.09.048
Reference: INS 15887

To appear in: *Information Sciences*

Received Date: 1 December 2018
Revised Date: 22 July 2020
Accepted Date: 19 September 2020

Distributed mining of time–faded heavy hitters

Marco Pulimeno, Italo Epicoco, Massimo Cafaro*

Dept. of Engineering for Innovation, University of Salento, Via per Monteroni, 73100 Lecce, Italy

Abstract

We present P2PTFHH (Peer–to–Peer Time–Faded Heavy Hitters) which, to the best of our knowledge, is the first distributed algorithm for mining time–faded heavy hitters on unstructured P2P networks. P2PTFHH is based on the FDCMSS (Forward Decay Count-Min Space-Saving) sequential algorithm, and efficiently exploits an averaging gossip protocol by merging in each interaction the involved peers' underlying data structures. We formally prove the convergence and correctness properties of our distributed algorithm and show that it is fast and simple to implement. Extensive experimental results, executed on both synthetic and real datasets, confirm that P2PTFHH retains the extreme accuracy and error bound provided by FDCMSS whilst showing excellent scalability. Our contributions are three-fold: (i) we prove that the averaging gossip protocol can be used jointly with our augmented sketch data structure for mining time–faded heavy hitters; (ii) we prove the error bounds on frequency estimation; (iii) we experimentally prove that P2PTFHH is extremely accurate and fast, allowing near real time processing of large datasets.

Keywords: peer–to–peer, averaging protocol, heavy hitters, time–fading model, sketches.

1. Introduction

Distributed algorithms supporting large scale decentralized systems are being studied and developed in order to cope with complex infrastructures (peer-to-peer and sensor networks, IoT (Internet of Things) devices) and to face the fast paced growing of both the volume and dimension of data, coupled with the increasing information flow deriving by novel applications in disparate fields (e.g., data mining, machine learning, artificial intelligence, optimisation and control, and social networking).

Truly distributed solutions (we exclude here the class of centralised algorithms in which an agent acts as a coordinator) provide the highly desirable properties of resilience and scalability, by empowering each individual agent (also known as a peer) with information sensing and decision making. However, designing a protocol in which the peers interact and collaborate towards a common goal can be quite challenging with regard to the communication network underlying the agents. One way to cope with this issue is to impose a well-known structured topology (e.g., a spanning-tree), whose properties can then be leveraged by the distributed protocol. A different approach, which shall be used in this paper, is the use of a gossip–based protocol [?]. In particular, we are interested in distributed averaging, which is a consensus algorithm serving as a foundational tool for the information dissemination of our distributed algorithm.

The task of distributed averaging is to drive the states of the nodes of a network towards an agreed common value with regard to the initial values held by the agents; this value is the average of the agents' initial values. In such a protocol, the agents share their status information only with a few other connected agents called neighbours.

A synchronous gossip–based protocol consists of a sequence of rounds in which each peer randomly selects one or more peers, exchanges its local state information with the selected peers and updates its local state

*Corresponding author.
Email addresses: marco.pulimeno@unisalento.it (Marco Pulimeno), italo.epicoco@unisalento.it (Italo Epicoco), massimo.cafaro@unisalento.it (Massimo Cafaro)

by using the received information. Therefore, gossip can be thought as a simple way of self-organising the peers' interactions in consensus seeking.

In this paper, we present our P2PTFHH (Peer–to–Peer Time–Faded Heavy Hitters) distributed algorithm. Building on distributed averaging, we design a gossip–based version of our sequential FDCMSS (Forward Decay Count-Min Space-Saving) algorithm for distributed mining of time–faded heavy hitters on unstructured P2P networks [?]. To the best of our knowledge, this is the first distributed protocol designed specifically for this task. The problem of mining heavy hitters (also known as frequent items) [? ?] is considered a fundamental data mining task and has been extensively studied. Among the many possible applications, we recall here its wide applicability in the context of the analysis of web query logs [?], network measurement, monitoring and traffic analysis [? ?], computational and theoretical linguistics [?]. In particular, we are interested to the problem of distributed mining of time–faded heavy hitters, a model that puts heavier weights on recent batches of streaming data than older batches [?].

Our contributions are three-fold: (i) we formally prove that the averaging gossip protocol can be used jointly with our augmented sketch data structure for mining time–faded heavy hitters; (ii) we prove the error bounds on frequency estimation; (iii) we experimentally prove that P2PTFHH is extremely accurate and fast, allowing near real time processing of large datasets.

This paper is organised as follows. We recall in Section 2 relevant related work and in Section 3 preliminary definitions and concepts that shall be used in the rest of the manuscript. We present in Section 4 our P2PTFHH algorithm and formally prove in Section 5 its correctness. Next, we provide extensive experimental results in Section 6. Finally, we draw our conclusions in Section 7.

2. Related Work

In this Section, we recall related work. We begin by discussing the literature related to sequential algorithms for heavy hitters, then we present relevant parallel algorithms and conclude with related work on distributed algorithms.

2.1. Sequential Algorithms for Heavy Hitters

Misra and Gries [?] designed in 1982 the first algorithm for mining heavy hitters. However, their solution was not proposed as a streaming algorithm, even though it works in a streaming context. The Lossy Counting and Sticky Sampling algorithms by Manku et al. [?], published in 2002, were considered the first streaming algorithms until 2003, the year in which the Misra and Gries algorithm was independently rediscovered and improved, with regard to its running time, by Demaine et al. (the so-called Frequent algorithm) [?] and Karp et al. [?]. The key idea to decrease the running time required to process an incoming item was the use of a better data structure, based on a hash table. A few years later, Metwally et al. presented Space-Saving, a novel algorithm providing a significant improvement with regard to the output's accuracy, whilst retaining the same running time and space required to process the input stream.

All of the previous algorithms for detecting heavy hitters are commonly known as counter–based, since they use a set of counters to keep track of the heavy hitters. Other algorithms are instead known as sketch–based, owing to their use of a sketch data structure to monitor the input stream. A sketch is usually a bi-dimensional array data structure containing a counter in each cell. Hash functions map the stream's items to corresponding cells in the sketch. In this class of algorithms, we recall here CountSketch by Charikar et al. [?], Group Test [?] and Count-Min [?] by Cormode and Muthukrishnan, hCount [?] by Jin et al. and CMSS [?] by Cafaro et al.

As an example of the possible applications of sketch data structures, we recall here their use in the context of Natural Language Processing (NLP). Obviously, NLP tasks require accurate statistics from large data. However, tracking large corpora to derive the required statistics is a memory intensive task. As a consequence, a recent research line focuses on the use of sketches. In particular, in [?] the authors show how to use the Count-Min sketch to build a single, unified model which is effective on the following important NLP tasks: (i) predicting the Semantic Orientation of words; (ii) distributional approaches for word similarity and (iii) unsupervised dependency parsing with little linguistics knowledge. Moreover, in [?

] the authors investigate ten different sketch based algorithms comparing the frequency estimation errors made.

Mining Correlated Heavy Hitters (CHHs) has been recently proposed by Lahiri et al. [?]; common applications requiring accurate mining of CHHs are related to network monitoring and management, and to anomaly and intrusion detection. For instance, taking into account the stream of pairs (source address, destination address) consisting of IP packets traversing a router, the data mining task is the identification of the nodes accounting for the majority of the traffic passing through that router (these are the heavy hitters over a single dimension); however, given a frequent source, it is important to simultaneously discover the identity of the destinations receiving the majority of connections by the same source. After mining the first dimension to detect the most important sources, the second dimension is mined to detect the frequent destinations in the context of each identified source, i.e., the stream's CHHs. Epicoco et al. [?] proposed a fast and more accurate algorithm for mining CHHs.

The above algorithms, when processing an input stream, do not discount the effect of old data. In practice, all of the items are given equal weight. However, this is not appropriate for those applications based on the underlying assumption that recent data is more useful and valuable than older, stale data. One way to handle such as a situation is using the so-called sliding window model [?] [?]. The key idea is the use of a temporal window to capture fresh, recent items. This window periodically slides forward, allowing detection of only those heavy hitters falling in the window.

The time–fading model [?] puts heavier weights on recent batches of streaming data than older batches. This is achieved by fading the frequency count of older items: each item in the stream has an associated timestamp that shall be used to determine its weight, by means of a decaying factor $0 < \lambda < 1$. Therefore, instead of an item's frequency count, the algorithms compute its decayed count (also known as decayed frequency) through a decay function assigning greater weight to more recent items. The older an item, the lower its decayed count is: in the case of exponential decay, the weight of an item occurred n time units in the past, is $e^{-\lambda n}$, which is an exponentially decreasing quantity.

Noteworthy algorithms for mining time–faded heavy hitters include λ-HCount [?] by Chen and Mei, FSSQ (Filtered Space-Saving with Quasi–heap) [?] by Wu et al. and the FDCMSS algorithm [?] [?] by Cafaro et al. These algorithms are sketch–based variants of Count-Min. Whilst λ-HCount and FSSQ rely on a backward decay exponential function, FDCMSS provides support for both backward and forward decay functions [?]. For instance, by using a polynomial forward function, FDCMSS provides more flexibility with regard to time fading, since the time fades more slowly than by using an exponential function. Moreover, the use of forward decay allows FDCMSS dealing with out of order arrival of stream's items easily. On the contrary, neither λ-HCount nor FSSQ can handle out of order items. Both λ-HCount and FSSQ require a dedicated data structure to keep track of frequent item candidates: λ-HCount uses a doubly linked list indexed by an hash function, whilst FSSQ uses a so–called Quasi-Heap, an heap in which heapify operations are delayed until necessary. The FDCMSS algorithm does not require additional space beyond its data structure, a sketch in which each cell hosts a Space-Saving stream summary holding exactly two counters. It has been proved that, fixing the same amount of space for all of the algorithms, FDCMSS outperforms λ-HCount and FSSQ with regard to the error committed. Moreover, FDCMSS provides the best error bound. Regarding the recall property (i.e., the ability of outputting all of the true heavy hitters), both λ-HCount and FSSQ guarantee 100% recall, whilst FDCMSS provides a probabilistic recall guarantee. Finally, with regard to the running time, FDCMSS is the fastest owing to its data structures and update strategy.

2.2. Parallel Algorithms for Heavy Hitters

Parallel algorithms designed for fast mining of heavy hitters span both the message–passing and shared–memory paradigms. Cafaro et al. [?] [?] [?] designed parallel versions of the Frequent and Space-Saving algorithms for message–passing architectures. Regarding shared-memory, we recall here parallel versions of Lossy Counting and Frequent by Zhang et al. [?] [?], parallel versions of Space-Saving by Dat et al. [?], Roy et al. [?], and Cafaro et al [?]. Parallel algorithms for fundamental frequency–based aggregates, including heavy hitters were designed by Tangwongsan et al. [?]. Among the accelerator based algorithms, exploiting GPU (Graphics Processing Unit) and the Intel Phi, we find Govindaraju et al. [?], Erra and

Frola [?] and Cafaro et al. [?] [?]. A parallel, message-passing based version of the CHHs algorithm [?] appears in Pulimeno et al. [?], whilst [?] presents a parallel message-passing based version of [?].

2.3. Distributed Algorithms for Heavy Hitters

Many algorithms for distributed mining of heavy hitters have been designed, e.g., [?], [?]. Among the gossip–based algorithms for mining heavy hitters on unstructured P2P networks, it is worth recalling here [?], [?], [?] and [?].

The algorithms by Sacha and Montresor [?] and by Aem and Azkasap [?] are based on the same ideas. Indeed, the dataset to be mined is partitioned among the peers, which periodically exchange their local state consisting of the current subset of the whole dataset. The key assumption is that a peer is allowed to store the whole dataset, even though this requires obviously a huge amount of space. The entire dataset is indeed obtained by means of the interactions in the distributed averaging gossip. The convergence of the averaging gossip protocol has been investigated in [?]; the authors proved that the variance around the mean value being computed is reduced in each round by a specific convergence factor.

Exchanging the full local state, besides requiring a huge amount of space, also increases the communication complexity. Therefore, in [?] the authors suggest to exchange only the top-k heavy hitters where k is a user's defined parameter. For each peer, the protocol's termination condition requires that the subset consisting of the top-k heavy hitters is stable, i.e., it does not change for a specified number of consecutive rounds. In [?], to cope with the communication complexity, the authors use an additional data structure. This is a hash table storing all of the items (these items are never deleted, increasing the space used), from which the algorithm randomly selects a specified number of items corresponding to a predefined message size. The termination condition requires two user's defined parameters: ϵ and *convLimit*. The algorithm terminates when for each item, the absolute difference between the estimated frequency in the current and previous rounds is less than or equal to ϵ for at least *convLimit* consecutive rounds.

Both [?] and [?] require space complexity linear in the length n of the dataset; this allows solving the exact problem rather than the approximate problem, but at the expense of space.

In Lahiri and Tirthapura [?], the authors design an algorithm based on random sampling of the items and the averaging gossip protocol. Each item is assigned a random weight in the interval (0, 1) and a peer maintains and exchanges in each round the t items whose weight is the lowest, with $t = \frac{128}{\psi^2} \ln \frac{3}{\delta}$; ψ and δ denote respectively a threshold and a probability of failure. The proposed approach can only detect heavy hitters, since the algorithm does not provide frequency estimation. In particular, the output is a list of items that with high probability (defined by δ) contains the heavy hitters (with regard to the ψ threshold). Regarding the space used, for each of the t items the algorithm stores a tuple consisting of four fields: the peer's identifier, the item's index in the peer's local dataset, the item's value and its random weight.

None of the previous gossip–based protocols has been designed to mine time–faded heavy hitters on unstructured P2P networks. To the best of our knowledge, P2PTFHH is the first distributed protocol designed specifically for this task.

3. Preliminary definitions

In this Section we introduce preliminary definitions and the notation used throughout the paper. In particular, Section 3.1 is related to time–faded heavy hitters, Section 3.2 recalls introductory concepts and definitions related to distributed systems whilst Section 3.3 introduces the Jelasity et al. averaging algorithm [?] that inspired our algorithm. In particular, we recall here their result related to the convergence factor of the uniform gossiping protocol.

3.1. Time–faded Heavy Hitters

We deal with an input data stream σ consisting of a sequence of n items drawn from a universe $\mathcal{U}$; without loss of generality, let m be the number of distinct items in $\mathcal{U}$ i.e., let $\mathcal{U} = \{1, 2, \ldots, m\}$, which we shall also denote by $[m]$. Let f_i be the frequency of the item $i \in \mathcal{U}$ (i.e., its number of occurrences in σ), and denote the frequency vector by $\mathrm{f} = (f_1, \ldots, f_m)$. Moreover, let $0 < \phi < 1$ be a support threshold, $0 < \epsilon < 1$ a tolerance such that $\epsilon < \phi$ and denote the 1-norm of f (which represents the total number of occurrences of all of the stream's items) by $||f||_1$.

In this paper we are concerned with the problem of detecting heavy hitters in a stream which is distributed among p peers, with the additional constraint that recent items are more relevant with regard to older ones. In the time–faded model, recent items are weighted more than former items while in the sliding window model only the items' occurrences which appear in a window sliding over time are considered. Our algorithm, P2PTFHH, works in the former model.

The time–fading model [?] [?] [?] [?] uses a fading frequency count of older items to emphasize the freshness of recent items. This is achieved by computing an item's decayed frequency through the use of a decay function that assigns greater weight to more recent occurrences of the item than to older ones: the older an occurrence, the lower its decayed weight is.

Definition 1. Let $w(t_i, t)$ be a decayed function which computes the decayed weight at time t for an occurrence of the item i arrived at time t_i. A decayed function must satisfy the following properties:

1. $w(t_i, t) = 1$ when $t_i = t$ and $0 \leq w(t_i, t) \leq 1$ for all $t > t_i$;
2. w is a monotone non-increasing function as time t increases, i.e., $t' \geq t \implies w(t_i, t') \leq w(t_i, t)$.

Related work has mostly exploited backward decay functions, in which the weight of an item is a function of its age, a, where the age at time $t > t_i$ is simply $a = t - t_i$. In this case, $w(t_i, t)$ is given by $w(t_i, t) = \frac{h(t-t_i)}{h(t-t)} = \frac{h(t-t_i)}{h(0)}$, where h is a positive monotone non-increasing function.

The term backward decay stems from the aim of measuring from the current time back to the item's timestamp. Prior algorithms and applications have been using backward exponential decay functions such as $h(a) = e^{-\lambda a}$, with $\lambda > 0$ being the decaying factor.

In our algorithm, we use instead a forward decay function, defined as follows (a detailed description of the forward decay approach is available in [?]). Under forward decay, the weight of an item is computed using the amount of time between the arrival of an item and a fixed point L, called the landmark time. By convention, the landmark time is earlier than the timestamps of all of the items. The idea is to look forward in time from the landmark to see an item, instead of looking backward from the current time.

Definition 2. Given a positive monotone non-decreasing function g, and a landmark time L, the forward decayed weight of an item i with arrival time $t_i > L$ measured at time $t \geq t_i$ is given by $w(t_i, t) = \frac{g(t_i-L)}{g(t-L)}$.

The denominator is used to normalise the decayed weight so that $w(t_i, t)$ is always less than or equal to 1 as requested by Definition 1.

Definition 3. The decayed frequency of an item v in the input stream σ, computed at time t, is given by the sum of the decayed weights of all the occurrences of v in σ: $f_v(t) = \sum_{v_i=v} w(t_i, t)$.

Definition 4. The decayed count at time t, $C(t)$, of a stream σ of n items is the sum of the decayed weights of all the items occurring in the stream: $C(t) = \sum_{i=1}^{n} w(t_i, t)$.

The Approximate Time–Faded Heavy Hitters (ATFHH) problem is formally stated as follows.

Problem 1. Approximate Time–Faded Heavy Hitters. Given a stream σ of items with an associated timestamp, a threshold $0 < \phi < 1$ and a tolerance $0 < \epsilon < 1$ such that $\epsilon < \phi$, and letting g be a decaying function used to determine the decayed frequencies and t be the query time, return the set of items H, so that:

- H contains all of the items v with decayed frequency at time t $f_v(t) > \phi C(t)$ (decayed heavy hitters);
- H does not contain any item v such that $f_v(t) \leq (\phi - \epsilon)C(t)$.

items	a	b	a	c	c	a	b	d	a	c
timestamp	1	2	3	4	5	6	7	8	9	10
weights	0.905	0.914	0.923	0.933	0.942	0.951	0.961	0.971	0.980	0.990

Table 1: Forward decayed weights for a sample stream

Example. Assuming the input stream $\sigma = \{a, b, a, c, c, a, b, d, a, c\}$ with associated discrete timestamps $\tau = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, and given a positive monotone non-decreasing function $g(t) = \lambda^t$, with $\lambda = 1.01$ and a landmark time $L = 1$, the forward decayed weights of the stream's items measured at query time $t = 11$ are reported in Table 1.

Regarding the decayed frequencies,

$$\begin{aligned} f_a &= 0.905 + 0.923 + 0.951 + 0.980 = 3.759 \\ f_b &= 0.914 + 0.961 = 1.875 \\ f_c &= 0.933 + 0.942 + 0.990 = 2.865 \\ f_d &= 0.971 \end{aligned} \tag{1}$$

In order to determine the time–faded heavy hitters, we need to compute the total decayed count $C(t)$ for $t = 11$, i.e., the query time. The total decayed count is simply the sum of the weights. In our example, $C(11) = 9.47$. The time–faded heavy hitters are finally determined by the ϕ parameter. Setting $\phi = 0.2$, the items a and c have decayed frequency greater than $\phi C(t) = 1.894$ hence they are time–faded heavy hitters.

In the following, when clear from the context, the query time shall be considered an implicit parameter, and we shall write f_v and C in place of $f_v(t)$ and $C(t)$. Our algorithm uses an averaging gossip protocol for unstructured P2P networks in which the peers exchange their internal state consisting of a Count-Min sketch data structure augmented by a Space-Saving summary associated to each sketch cell. In the following, we recall the main properties of the Count-Min and Space-Saving algorithms for ordinary, non decaying frequencies and the averaging gossip protocol.

Count-Min is based on a sketch of $d \times w$ cells, where each item is mapped to by means of d different pairwise independent hash functions. Count-Min solves the frequency estimation problem for arbitrary items with an estimation error $\epsilon \leq e/w$ and with probability $1 - e^{-d}$. The algorithm may also be extended to solve the Approximate Heavy Hitters problem as well, by using an additional heap data structure which is updated each time a cell is updated. Since in Count-Min the frequencies stored in the cells overestimate the true frequencies, a point query for an arbitrary item simply inspects all of the d cells in which the query item is mapped to by the corresponding hash functions and returns the minimum of those d counters.

Space-Saving is a counter-based algorithm solving the heavy hitters problem. It is based on a stream summary data structure consisting of a given number of counters $k \ll n$, n being the length of the stream. Each counter monitors an item in the stream and tracks its frequency. A substitution strategy is used when the algorithm processes an item not already monitored and all of the counters are occupied.

Let σ be the input stream and denote by $\mathcal{S}$ the summary data structure of k counters used by the Space-Saving algorithm. Moreover, denote by $|\mathcal{S}|$ the sum of the counters in $\mathcal{S}$, by f_v the exact frequency of an item v and by $\hat{f}_v$ its estimated frequency, let $\hat{f}_{min}$ be the minimum frequency in $\mathcal{S}$. If there exist at least one counter not monitoring any item, $\hat{f}_{min}$ is zero.

Finally, denote by $\mathrm{f} = (f_1, \ldots, f_m)$ the frequency vector. The following relations hold (as shown in [?]):

$$|\mathcal{S}| = \|\mathrm{f}\|_1, \tag{2}$$

$$\hat{f}_v - f_v \leq \hat{f}_{min} \leq \left\lfloor \frac{\|\mathrm{f}\|_1}{k} \right\rfloor, \quad v \in \mathcal{U}. \tag{3}$$

3.2. Distributed Systems and Gossip–based Protocols

A distributed system is a collection of computing units, each one abstracted through the notion of a process. The processes are assumed to work cooperatively towards a common goal. In order to achieve such a goal, they exchange information through messages.

The set of processes is usually assumed to be static (even though in certain situations, the set of processes may be dynamic). We denote the set of p distinct processes by $V = \{p_1, p_2, \dots, p_p\}$. Each process p_i is sequential, i.e., it can execute one step at a time. The processes may communicate by sending and receiving messages through a set of channels E. Each channel $e = (p_i, p_j)$ is assumed to be reliable (it does not create, modify, or duplicate messages), bidirectional (can carry messages in both directions) and to have an infinite capacity (can contain any number of messages, each of any size).

From a structural point of view, a distributed system can be represented by a connected undirected graph $G = (V, E)$. Each process p_i has a set of neighbours, denoted $neighbours_i$. For a process p_i, $neighbours_i = \{p_j : (p_i, p_j) \in E\}$.

A distributed algorithm is a collection of n automata, one per process. An automaton formally describes a finite state machine along with its transitions, corresponding to the sequence of steps executed by the corresponding process. Moreover, an automaton is allowed to send a message on a channel or receive a message on any channel. The corresponding operations are send() and receive().

A distributed synchronous algorithm is an algorithm which has been designed for execution on a synchronous distributed system. The progress of such a system is governed by an external global clock, and the processes, which are equipped with local clocks, collectively execute a sequence of rounds, each round corresponding to a tick of the global clock. In synchronous distributed systems there is a known upper bound on the relative speed of process' execution, on message transmission delays and on the drift rates of local clocks with respect to the global clock.

During a round, a process sends at most one message to each of its neighbours. In synchronous systems a message sent by a process during a round r is received by its destination process during the same round r. As a consequence, when a process proceeds to round $r + 1$, it has received and processed all of the messages which have been sent to it during round r.

The gossip–based protocol [?] is a synchronous distributed algorithm consisting of periodic rounds. In each of the rounds, a process, which is known in this context as a peer (or agent) randomly selects one or more of its neighbours, exchanges its local state with them and finally updates its local state. The information is disseminated through the network by using one of the following possible communication methods: (i) push, (ii) pull or (iii) push–pull. The main difference between push and pull is that in the former a peer randomly selects the peers to whom it wants to send its local state, whilst in the latter it randomly selects the peers from whom to receive the local state. Finally, in the hybrid push–pull communication style, a peer randomly selects the peers to send to and from whom to receive the local state. In this synchronous distributed model it is assumed that updating the local state of a peer is done in constant time, i.e., with $O(1)$ worst-case time complexity; moreover, the duration of a round is such that each peer can complete a push–pull communication within the round. Finally, the performance of a gossip–based protocol is measured as the number of rounds required to achieve convergence.

We are interested in a specific gossip–based protocol, which is called distributed averaging, and can be considered as a consensus protocol. For the purpose of our theoretical analysis, we assume that peers and communication links do not fail, and that neither new peers can join the network nor existing peers can leave it (the so-called churning phenomenon). Therefore, the graph $G = (V, E)$ in which V is the set of peers and E models the connectivity (fully describing the underlying network topology), is not time-varying. However, it is worth noting here that our algorithm also works in time-varying graphs in which the network can change owing to failures or churning and we shall show an experimental evidence of that in Section 6.3, in which we discuss the effect of churn.

In the sequel, we shall simplify the notation by using the index i of a peer to denote the corresponding process p_i. Consequently, a channel e between the peers i and j shall be denoted by (i, j).

In uniform gossiping, a peer i can communicate with a randomly selected peer j. Instead, in our scenario the communication among the peers is restricted to neighbour peers, i.e., two peers i and j are allowed to communicate if and only if a channel - i.e., a one-hop communication link - exists between them; we assume that communication links are bidirectional. Initially, each peer i is provided with or computes a real number v_i; the distributed averaging problem requires designing a distributed algorithm allowing each peer to compute the average $v_{\text{avg}} = \frac{1}{p} \sum_{i=1}^{p} v_i$ by exchanging information only with its neighbours.

Letting $v_i(r)$ be the peer i estimated value of v_{avg} at round r, a gossip interaction between peers i and j updates both peers' variables so that at round $r + 1$ it holds that $v_i(r + 1) = v_j(r + 1) = \frac{1}{2}(v_i(r) + v_j(r))$. Of course, for a peer i which is not gossiping at round r it holds that $v_i(r + 1) = v_i(r)$. It can be shown that distributed averaging converges exponentially fast to the target value v_{avg}. In general, a peer is allowed to gossip with at most one peer at a time. In our algorithm, we allow each peer the possibility of gossiping with a predefined number of neighbours. We call fan-out of the peer i the number of its neighbours with which it communicates in each round; therefore, $1 \leq$ fan-out $\leq |\{j : (i, j) \in E\}|$. Therefore, we explicitly allow two or more pairs of peers to gossip at the same time, with the constraint that the pairs have no peer in common. We formalise this notion in the following definition.

Definition 5. Two gossip pairs of peers (i, j) and (x, y) are noninteracting if neither i nor j equals either x or y.

In our algorithm multiple noninteracting pairs of allowable gossips may occur simultaneously. Non-interactivity is required in order to preserve and guarantee the correctness of the results; in the literature non-interactivity is also called atomic push–pull communication: given two peers i and j, if peer i sends a push message to j, then peer i can not receive in the same round any intervening push message from any other peer k before receiving the pull message from j corresponding to its initial push message.

It is worth noting here that our algorithm does not require explicitly assigning identifiers to the peers, and we do so only for convenience, in order to simplify the analysis; however, we do assume that each peer can distinguish its neighbours.

3.3. Jelasity's averaging algorithm

In the work by Jelasity et al. [?], the proposed averaging protocol is interpreted as a distributed variance reduction algorithm, fully equivalent to a centralised algorithm which computes the average value among a set of elements. Consider a variance measure σ_r^2 defined as:

$$\sigma_r^2 = \frac{1}{p-1} \sum_{l=1}^{p} (w_{r,l} - \bar{w})^2 , \tag{4}$$

where $w_{r,l}$ is the value held by the peer l after r rounds of the gossip algorithm and $\bar{w} = \frac{1}{p} \sum_{l=1}^{p} w_{0,l}$ is the mean of the initial values held by the peers. The authors in [?] state that if ψ_l is a random variable denoting the number of times a node l is chosen as a member of the pair of nodes exchanging their states during a round of the protocol, and each pair of nodes is uncorrelated, then the following theorem holds.

Theorem 1. [?] If:

1. the random variables $\psi_1, \ldots, \psi_p$ are identically distributed (let ψ denotes a random variable with this common distribution);
2. after a pair of nodes (i, j) is selected, the number of times the nodes i and j shall be selected again have identical distributions;

then:

$$\mathbb{E}[\sigma_{r+1}^2] \approx \mathbb{E}[2^{-\psi}]\mathbb{E}[\sigma_r^2]. \tag{5}$$

The random variable ψ only depends on how the pair of nodes are selected. From equation (5), the convergence factor is defined as:

$$\frac{\mathbb{E}[\sigma_{r+1}^2]}{\mathbb{E}[\sigma_r^2]} = \mathbb{E}[2^{-\psi}]; \tag{6}$$

Therefore, the convergence factor depends on ψ and, as a consequence, on the pair selection method. Jelasity et al. compute the convergence factor for different methods of pair selection; we are only interested in the one which allows simulating the distributed gossip–based averaging protocol. This method consists in drawing a random permutation of the nodes and then, for each node in that permutation, choosing another random node in order to form a pair. For this selection method, the convergence factor is $\gamma = \mathbb{E}[2^{-\psi}] = 1/(2\sqrt{e})$.

4. The algorithm

In this Section, we start by recalling our sequential algorithm FDCMSS [?]. The key data structure is an augmented Count-Min sketch $\mathcal{D}$, whose dimensions d (rows) and w (columns) are derived by the input parameters ϵ, the error tolerance, and δ, the probability of failure. Whilst every cell in an ordinary Count-Min sketch contains a counter used for frequency estimation, in our case a cell holds a Space-Saving summary with exactly two counters. The idea behind the augmented sketch is to monitor the time–faded items that the sketch's hash functions map to the corresponding cells by an instance of Space-Saving with two counters, so that for a given cell we are able to determine a majority item candidate with regard to the sub-stream of items falling in that cell. In fact, we proved that, with high probability, if an item is frequent, then it appears as a majority item candidate in at least one of the d sketch cells in which it falls.

Indeed, by using a Space-Saving summary with two counters in each cell, and letting $C_{i,j}$ denote the total decayed count of the items falling in the cell $\mathcal{D}[i][j]$, the majority item is, if it exists, the item whose decayed frequency is greater than $\frac{C_{i,j}}{2}$. The corresponding majority item candidate in the cell is the item monitored by the Space-Saving counter whose estimated decayed frequency is maximum.

The main idea of the distributed version of the FDCMSS algorithm is to let each peer process its local data stream with the FDCMSS algorithm. Then, the peers engage in a gossip–based distributed averaging protocol, exchanging their local state which consists of the augmented sketch data structure obtained after processing the input stream and an estimate of the number of peers in the network.

The algorithm's initialisation (pseudo-code provided as Algorithm 1) requires as input parameters d and w, the dimensions of the local sketch; $\mathcal{N}_l$, the local dataset to be processed, being l the peer's identifier.

The initialisation procedure starts calling the CreateSketch function which allocates and returns an augmented sketch: for each one of the $d \times w$ cells of the sketch, CreateSketch allocates a Space-Saving summary with two counters c_1 and c_2, both maintaining an item and a frequency value. Given a counter c_j, $j = 1, 2$, we denote by $c_j.i$ and $c_j.f$ respectively the counter's item and its estimated decayed frequency. CreateSketch selects d pairwise independent hash functions $h_1, \ldots, h_d : [m] \rightarrow [w]$, mapping m distinct items into w cells. It is worth noting here that each peer is required to use the same d hash functions in order to guarantee the correctness of the results.

After the main data structures are created, the peer whose identifier is $l = 1$ sets $\tilde{q}_{0,l}$ to 1 and all of the other peers sets this value to zero. The variable $\tilde{q}_{0,l}$ represents the estimate of peer l at round 0 of the number p of peers (to be obtained by using the distributed averaging protocol: indeed, upon convergence this value approaches $1/p$ with high probability).

Algorithm 1 P2PTFHH: Initialisation

```
Require: $d, w$, number of rows and columns of the sketch; $\mathcal{N}_l$, local dataset to be processed.
Ensure: Initialisation of the internal peer's state.
 1: procedure Initialise($d$, $w$, $\mathcal{N}_l$)                ▷ initialisation of node $l$
 2:     Let $l$ be the peer's identifier
 3:     $\mathcal{D}_{0,l} \leftarrow$ CreateSketch($w$,$d$)
 4:     if $l == 1$ then
 5:         $\tilde{q}_{0,l} \leftarrow 1$
 6:     else
 7:         $\tilde{q}_{0,l} \leftarrow 0$
 8:     end if
 9:     FDCMSS_UPDATE( $\mathcal{D}_{0,l}$, $\mathcal{N}_l$)
10:     $state_{0,l} \leftarrow (\mathcal{D}_{0,l}, \tilde{q}_{0,l})$
11: end procedure
```

Each peer processes its local data stream following the update procedure of the FDCMSS algorithm, shown in pseudo-code as Algorithm 2. For each item i with timestamp t_i, the non normalised forward decayed weight x of the item is computed, then the d cells in which the item is mapped to by the corresponding hash functions $h_j(x)$, $j = 1, \ldots, d$ are updated by using the Space-Saving item update procedure.

We shortly describe here the update procedure of the Space-Saving algorithm, SpaceSavingUpdate in Algorithm 2. Let $\mathcal{S}$ denote the Space-Saving summary with two counters corresponding to the cell to be updated. When processing an item which is already monitored by a counter, its estimated frequency is incremented by the non normalised weight x. When processing an item which is not already monitored by one of the available counters, there are two possibilities. If a counter is available, it will be in charge of monitoring the item, and its estimated frequency is set to the non normalised weight x. Otherwise, if all of the counters are already occupied (their frequencies are different from zero), the counter storing the item with minimum frequency is incremented by the non normalised weight x. Then, the monitored item is evicted from the counter and replaced by the new item. This happens since an item which is not monitored can not have a frequency greater than the minimal frequency.

Algorithm 2 FDCMSS-UPDATE

```
Require: $\mathcal{D}_{0,l}$, sketch data structure of peer $l$; $\mathcal{N}_l$, local dataset to be processed.
Ensure: update of sketch related to items in $\mathcal{N}_l$.
1: procedure FDCMSS_UPDATE($\mathcal{D}_{0,l}$, $\mathcal{N}_l$)        ▷ let $i$ be an item, and $t_i$ its timestamp
2:     for each $(i, t_i) \in \mathcal{N}_l$ do
3:         $x \leftarrow g(t_i - L)$        ▷ compute the non normalised decayed weight of item $i$
4:         for $j = 1$ to $d$ do
5:             $\mathcal{S} \leftarrow \mathcal{D}_{0,l}[j][h_j(i)]$
6:             SpaceSavingUpdate($\mathcal{S}, i, x$)        ▷ update the sketch
7:         end for
8:     end for
9: end procedure
```

We represent the local state of a peer l at round r as a tuple $state_{r,l}$ consisting of the peer's local sketch $\mathcal{D}_{r,l}$, and the estimate $\tilde{q}_{r,l}$.

In the P2PTFHH algorithm, all of the peers participate to the distributed gossip protocol for R rounds (Gossip procedure in Algorithm 3). During each round, a peer increments r, the rounds counter, selects fan-out neighbours uniformly at random and sends to each of them its local state in a message of type push. Upon receiving a message, each peer executes the on_receive procedure, whose pseudo-code is shown as Algorithm 3. From the message, the peer extracts the message's type (which can be 'push' or 'pull'), sender and state sent. A message is processed accordingly to its type as follows. A push message is handled in two steps. In the first one, the peer updates its local state by using the state received; this is done by invoking the update procedure that we shall describe later. In the second one, the peer sends back to the sender, in a message of type pull, its updated local state. A pull message is handled by a peer setting its local state equal to the state received.

The update procedure, which is the last procedure used by Algorithm 3, works as follows: the two local sketches $\mathcal{D}_i$ and $\mathcal{D}_j$ belonging respectively to the peers i and j are merged by invoking the merge procedure, producing the new sketch $\mathcal{D}$; finally, we compute as required by the averaging protocol the estimate $\tilde{q}$ and return the updated state just computed.

Two sketches are merged by the merge procedure (pseudo-code provided as Algorithm 4) as follows: for every corresponding cell in the two sketches to be merged, the hosted Space-Saving summaries are merged following the steps described in [?], i.e., building a temporary summary $\mathcal{S}_C$ consisting of all of the items monitored by both $\mathcal{S}_1$ and $\mathcal{S}_2$. To each item in $\mathcal{S}_C$ is assigned a decayed frequency computed as follows: if an item is present in both $\mathcal{S}_1$ and $\mathcal{S}_2$, its frequency is the sum of its corresponding frequencies in each summary; if the item is present only in one of either $\mathcal{S}_1$ or $\mathcal{S}_2$, its frequency is incremented by the minimum frequency of the other summary – moreover, since we want to implement a distributed averaging protocol, we here slightly modify the merge procedure in [?], dividing by two the computed frequency. At last, in order to derive the merged summary, we take only the two items in $\mathcal{S}_C$ with the greatest frequencies and discard the others.

Once R rounds of gossip have been executed, the user can invoke the query procedure (pseudo-code

Algorithm 3 P2PTFHH: Gossip protocol

```
1: procedure GOSSIP
2:     for $r = 0$ to $R$ do
3:         $neighbours \leftarrow$ select fan-out random neighbours
4:         for each $i \in neighbours$ do
5:             SEND($push$, $i$, $state_{r,l}$)
6:         end for
7:     end for
8: end procedure
9: procedure ON_RECEIVE($msg$)
10:    if $msg.type == push$ then
11:        $state_{r+1,l} \leftarrow$ UPDATE($msg.state$, $state_{r,l}$)
12:        SEND($pull$, $msg.sender$, $state_{r+1,l}$)
13:    end if
14:    if $msg.type == pull$ then
15:        $state_{r+1,l} \leftarrow msg.state$
16:    end if
17: end procedure
18: procedure UPDATE($state_i$, $state_j$)
19:    $(\mathcal{D}_i, \tilde{q}_i) \leftarrow state_i$; $(\mathcal{D}_j, \tilde{q}_j) \leftarrow state_j$
20:    $\mathcal{D} \leftarrow$ MERGE($\mathcal{D}_i$, $\mathcal{D}_j$)
21:    $\tilde{q} \leftarrow \frac{\tilde{q}_i + \tilde{q}_j}{2}$
22:    $state \leftarrow (\mathcal{D}, \tilde{q})$
23:    return $state$
24: end procedure
```

Algorithm 4 Merge

```
Require: $\mathcal{D}_1, \mathcal{D}_2$, sketch data structures to be merged.
Ensure: $\mathcal{G}$, merged sketch.
1: procedure MERGE($\mathcal{D}_1, \mathcal{D}_2$)
2:     for each $\mathcal{S}^1_{ij} \in \mathcal{D}_1, \mathcal{S}^2_{ij} \in \mathcal{D}_2$ do
3:         $m_1 \leftarrow \min(c^1_1.f, c^1_2.f)$; $m_2 \leftarrow \min(c^2_1.f, c^2_2.f)$    ▷ $m_1$,$m_2$, the minimum of counters' frequency in $\mathcal{S}^1_{ij}$, $\mathcal{S}^2_{ij}$
4:         for each $c_{s_1} \in \mathcal{S}_1$ do
5:             $c_{s_2} \leftarrow$ Extract($\mathcal{S}_2, c_{s_1}.i$)
           If $c_{s_2}$ then $c_{s_c}.f \leftarrow \frac{c_{s_1}.f + c_{s_2}.f}{2}$ else $c_{s_c}.f \leftarrow \frac{c_{s_1}.f + m_2}{2}$
6:             $c_{s_c}.i \leftarrow c_{s_1}.i$
7:             Insert($\mathcal{S}_C, c_{s_c}$)
8:         end for
9:         for each $c_{s_2} \in \mathcal{S}_2$ do
10:            $c_{s_c}.i \leftarrow c_{s_2}.i$
11:            $c_{s_c}.f \leftarrow \frac{c_{s_2}.f + m_1}{2}$
12:            Insert($\mathcal{S}_C, c_{s_c}$)
13:        end for
14:        Purge($\mathcal{S}_C$)    ▷ $\mathcal{S}_C$ now contains 2 counters with the greatest frequencies
15:        $\mathcal{G}[i][j] \leftarrow \mathcal{S}_C$
16:    end for
17:    return $\mathcal{G}$
18: end procedure
```

Algorithm 5 QUERY

```
Require: $t$, query time.
Ensure: $H$, the set of heavy hitters.
procedure QUERY($t$)
    $(\tilde{\mathcal{D}}_{r,l}, \tilde{q}_{r,l}) \leftarrow state_{r,l}$
    $\epsilon^* \leftarrow p^* \times \sqrt{\frac{\gamma^r}{\delta_g}}$
    $\tilde{p}_{r,l} \leftarrow 1/\tilde{q}_{r,l}$
    $\tilde{C}_{r,l} \leftarrow 0$
    for each summary $\mathcal{S} \in \tilde{\mathcal{D}}_{r,l}$ do,
        for each counter $c \in \mathcal{S}$ do
            $\tilde{C}_{r,l} \leftarrow \tilde{C}_{r,l} + c.f$
        end for
    end for
    $\tilde{C}_{r,l} \leftarrow \frac{\tilde{C}_{r,l}}{g(t-L)}$
    $H \leftarrow \emptyset$
    $\tau \leftarrow \phi \tilde{C}_{r,l} \frac{1-\epsilon^*}{1+\epsilon^*}$
    for each summary $\mathcal{S} \in \tilde{\mathcal{D}}_{r,l}$ do
        let $c_1$ and $c_2$ be the counters in $\mathcal{S}$
        $c_m \leftarrow \text{argmax}(c_1, c_2)$          ▷ $c_m$ the counter with maximum decayed count
        if $\frac{c_m.f}{g(t-L)} > \tau$ then
            $p \leftarrow \text{PointEstimate}(c_m.i, t)$
            if $p > \tau$ then
                $H \leftarrow H \cup \{(c_m.i, p)\}$
            end if
        end if
    end for
    return $H$
end procedure
procedure PointEstimate($i, t$)
    $answer \leftarrow \infty$
    for $k = 1$ to $d$ do
        $\mathcal{S} \leftarrow \tilde{\mathcal{D}}_{r,l}[k][h_k(i)]$          ▷ let $c_1$ and $c_2$ be the counters in $\mathcal{S}$
        if $i == c_1.i$ then
            $answer \leftarrow \min(answer, c_1.f)$
        else if $i == c_2.i$ then
            $answer \leftarrow \min(answer, c_2.f)$
        else
            $m \leftarrow \min(c_1.f, c_2.f)$
            $answer \leftarrow \min(answer, m)$
        end if
    end for
    return $\frac{answer}{g(t-L)}$
end procedure
```

provided as Algorithm 5) on an arbitrary peer to retrieve the time–faded heavy hitters. This is done by computing τ, a threshold that determines whether an item is a candidate heavy hitter or not, $\tilde{p}_{r,l}$, the estimate of p and $\tilde{C}_{r,l}$, the global decayed count. Note that τ is defined in terms of ϵ^*, whose meaning shall be explained in Section 5.3. The total decayed count can be computed by summing up the decayed frequencies stored by the Space-Saving counters inside each single cell in a row of the sketch data structure $\mathcal{D}$. It is worth noting here that the decayed frequencies are still non normalised; the normalisation occurs dividing by $g(t-L)$, where t is the query time and L denotes a landmark time. The query procedure initialises H, an empty set, and then it inspects each of the $d \times w$ cells in the sketch $\mathcal{D}$. For a given cell, we determine c_m, the counter in the data structure $\mathcal{S}$ with maximum decayed count. We compare the normalised decayed count stored in c_m with the threshold $\tau = \phi \tilde{C}_{r,l} \frac{1-\epsilon^*}{1+\epsilon^*}$. If the normalised decayed frequency is greater, we pose a point query for the item $c_m.i$, shown in pseudo-code as the PointEstimate procedure. If the returned value is greater than the threshold τ, then we insert in H the pair $(c_m.i, p)$.

The point query for an item j returns its estimated decayed frequency. We inspect each of the d cells in which the item j is mapped to by the corresponding hash functions in order to determine the minimum decayed frequency of the item; for those cells where the item is not monitored, the minimum decayed frequency stored in the counters for that cell is considered. Since the frequencies stored in all of the counters of the sketch are not normalised, we return the normalised frequency answer dividing by $g(t-L)$. At the end of the query procedure the set H is returned.

5. Correctness

In this Section, we prove that in our algorithm every peer correctly converges to a suitable sketch data structure that can be queried to solve the Approximate Time–Faded Heavy Hitters problem. We also prove a bound on the frequency estimation error committed by our algorithm.

Before delving into the details of the analysis, we recall in Section 5.1 the main properties exhibited by the FDCMSS algorithm and by the Merge procedure (Algorithm 4) for merging FDCMSS sketches [?], that we use in Algorithm 3. Next, we show in Section 5.2 that P2PTFHH converges and in Section 5.3 we prove the error bound due to the gossip-based approximation. These results are then used to formally prove the correctness of the algorithm in Section 5.4. We also provide a useful error bound on the frequency estimation in Section 5.5, and provide the insights for fine tuning of the algorithm's parameters in Section 5.6.

5.1. FDCMSS algorithm and merging of augmented sketches

We recall here the main results related to the theoretical properties of the FDCMSS algorithm. Cafaro et al. in [?] proved that FDCMSS solves the Approximate Time–Faded Heavy Hitters problem and provided a bound on its frequency estimation error. In particular, the authors state that, with high probability, if a time–faded item is frequent, then, in at least one of the FDCMSS sketch cells where it is mapped to by the hash functions, it is a majority item with regard to the sub-stream of items falling in the same cell. Therefore, FDCMSS shall detect it.

The following theorem holds, being $d \times w$ the dimensions of the sketch and ϕ the frequent threshold.

Theorem 2. [?] If an item i is frequent, then it appears as a majority item candidate in at least one of the d cells in which it falls, with probability greater than or equal to $1 - (\frac{1}{2\phi w})^d$.

Moreover, regarding the error bound on frequency estimation of the FDCMSS algorithm, if f_i is the exact decayed frequency of item i in the stream σ and $\hat{f}_i$ is the estimated decayed frequency of item i returned by FDCMSS and C is the total decayed count of all of the items in the input stream, then, the authors proved that the following error bound holds:

Theorem 3. [?] Denoting by C the total decayed count, and by d and w the number of rows and columns of the augmented sketch data structure, then $\forall i \in [m]$, $\hat{f}_i$ estimates the exact decayed count f_i of item i at query time with error less than $eC/2w$ and probability greater than $1 - e^{-d}$, i.e.:

$$f_i \le \hat{f}_i \le f_i + \frac{eC}{2w}; \tag{7}$$

P2PTFHH follows the same structure of the gossip–based averaging protocol by Jelasity et al., where, instead of single values, two communicating nodes provide two sketches to be merged and averaged. The merge operation refers to the procedure introduced by Cafaro et al. in [?], where the authors proved that the merge procedure of two augmented sketch data structures preserves all of the properties of the sketch, including the fact that an FDCMSS sketch is 1-norm equivalent to a classical Count-Min sketch for the same input stream, i.e. the value of a cell in the Count-Min sketch is equal to the sum of the Space-Saving counters in the same cell of the FDCMSS sketch.

Therefore, if $\mathcal{D}_1$ and $\mathcal{D}_2$ are FDCMSS sketches which respectively refer to the streams $\mathcal{N}_1$ and $\mathcal{N}_2$, then the sketch $\mathcal{D} = \mathcal{D}_1 \oplus \mathcal{D}_2$ is an FDCMSS sketch for the stream $\mathcal{N} = \mathcal{N}_1 \uplus \mathcal{N}_2$, where we have used the symbol $\oplus$ to indicate the merge operation.

5.2. Convergence of P2PTFHH

In order to prove the convergence of P2PTFHH, we shall use multisets to represent the input streams and introduce some operations on multisets and sketches in order to replicate the averaging pattern of Jelasty et al. algorithm.

Definition 6. A multiset $\mathcal{N} = (N, f_{\mathcal{N}})$ is a pair where N is some set, called the underlying set of elements, and $f_{\mathcal{N}} : N \to \mathbb{N}$ is a function. The generalised indicator function of $\mathcal{N}$ is

$$I_{\mathcal{N}}(x) := \begin{cases} f_{\mathcal{N}}(x) & x \in N, \\ 0 & x \notin N, \end{cases} \tag{8}$$

where the integer–valued function $f_{\mathcal{N}}$, for each $x \in N$, provides its frequency (or multiplicity), i.e., the number of occurrences of x in $\mathcal{N}$. The cardinality of $\mathcal{N}$ is expressed by

$$|\mathcal{N}| := Card(\mathcal{N}) = \sum_{x \in N} I_{\mathcal{N}}(x), \tag{9}$$

whilst the cardinality of the underlying set N is

$$|N| := Card(N) = \sum_{x \in N} 1. \tag{10}$$

A multiset, or bag, is defined by a proper set (the support set) and a multiplicity function: it is a set where elements can be repeated, i.e., an element in a multiset can have multiplicity greater than one.

For our specific application, we relax the above definition of multiset, allowing the multiplicity function f_N to assume real values. Let $\mathcal{M}$ be the class of all of the multisets with support set included in $\mathcal{U}$. We introduce the operation $\oslash_v : \mathcal{M} \to \mathcal{M}$, so that $\oslash_v(\mathcal{N}) = (N, f_N/v))$, i.e, the multiset $\oslash_v(\mathcal{N})$ has the same support set of $\mathcal{N}$, but each element has a fraction $1/v$ of the multiplicity that it has in $\mathcal{N}$. When $v \in \mathbb{N}$, we have that $\biguplus_{i=1}^{v} \oslash_v(\mathcal{N}) = \oslash_{\frac{1}{v}}(\oslash_v(\mathcal{N})) = \mathcal{N}$.

We can define a similar operation for an augmented FDCMSS sketch $\mathcal{D}$ and, by abusing the same symbol, we say that $\oslash_v(\mathcal{D}) = \mathcal{D}'$, where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by dividing all of the frequencies' values in $\mathcal{D}$ by v.

It is immediate to see that if $\mathcal{D}$ is an augmented sketch for $\mathcal{N}$, then $\oslash_v(\mathcal{D})$ is a augmented sketch for $\oslash_v(\mathcal{N})$ and theorems 2 and 3 continue to hold.

Algorithm 6, called AVG-Merge, is a simplified centralised version of our distributed P2PTFHH algorithm. AVG-Merge simulates the distributed averaging protocol, operating on the global state of the network and allowing us to simplify the theoretical analysis. GetPair, not reported here, is a procedure which selects a pair of nodes and meets the properties required by Theorem 1.

Initially, each peer computes a local augmented sketch using its input stream, through the execution of the FDCMSS algorithm, then the distributed protocol starts. The initial distributed state of the system

Algorithm 6 AVG-Merge: centralised merging of FDCMSS sketches

```
Require: $\boldsymbol{\mathcal{D}}_r = (\mathcal{D}_{r,1}, \mathcal{D}_{r,2}, \ldots, \mathcal{D}_{r,p})$, the vector of FDCMSS sketches at round $r$; $p$, the number of peers.
Ensure: $\boldsymbol{\mathcal{D}}_{r+1}$, the vector of FDCMSS sketches updated for round $r + 1$.
  procedure AVG_Merge($\boldsymbol{\mathcal{D}}_r$, $p$)
      $l \leftarrow 0$
      while $l < p$ do
          $(i, j) \leftarrow$ GetPair( )
          $\mathcal{D}_{r,i} \leftarrow \mathcal{D}_{r,j} \leftarrow \oslash_2(\mathcal{D}_{r,i} \oplus \mathcal{D}_{r,j})$
          $l \leftarrow l + 1$
      end while
      $\boldsymbol{\mathcal{D}}_{r+1} \leftarrow \boldsymbol{\mathcal{D}}_r$
      return $\boldsymbol{\mathcal{D}}_{r+1}$
  end procedure
```

can be represented by the vector of the local augmented sketches $\boldsymbol{\mathcal{D}}_0 = (\mathcal{D}_{0,1}, \mathcal{D}_{0,2}, \ldots, \mathcal{D}_{0,p})$, where p is the number of peers participating in the protocol. Another vector is naturally associated to $\boldsymbol{\mathcal{D}}_0$: the vector of the local input streams $\boldsymbol{\mathcal{N}}_0 = (\mathcal{N}_{0,1}, \mathcal{N}_{0,2}, \ldots, \mathcal{N}_{0,p})$. It holds that $\biguplus_{l=1}^{p} \mathcal{N}_{0,l} = \mathcal{N}$, where we denote by $\mathcal{N}$ the global input stream.

Each call to AVG-Merge corresponds to a round of P2PTFHH. It modifies $\boldsymbol{\mathcal{D}}_r$, the vector of the local sketches held by the peers at the end of round r, producing the vector $\boldsymbol{\mathcal{D}}_{r+1}$. Furthermore, implicitly also $\boldsymbol{\mathcal{N}}_r$, the vector of local input streams to which the sketches refer, changes to $\boldsymbol{\mathcal{N}}_{r+1}$. Indeed, let $\boldsymbol{\mathcal{D}}_r$ and $\boldsymbol{\mathcal{N}}_r$ be respectively the vectors of the sketches owned by the peers and the corresponding partition of the input stream $\mathcal{N}$ after the rth round. Then, after each iteration of the main loop of AVG-Merge, letting (i, j) be the pair of communicating peers, i.e. the pair selected by GetPair, the vector of sketches becomes:

$$\boldsymbol{\mathcal{D}}'_r = (\mathcal{D}_{r,1}, \mathcal{D}_{r,2}, \ldots, \oslash_2(\mathcal{D}_{r,i} \oplus \mathcal{D}_{r,j}), \ldots, \oslash_2(\mathcal{D}_{r,i} \oplus \mathcal{D}_{r,j}), \ldots, \mathcal{D}_{r,p}), \tag{11}$$

and the corresponding vector of partitions of the input stream shall change to:

$$\boldsymbol{\mathcal{N}}'_r = (\mathcal{N}_{r,1}, \mathcal{N}_{r,2}, \ldots, \oslash_2(\mathcal{N}_{r,i} \uplus \mathcal{N}_{r,j}), \ldots, \oslash_2(\mathcal{N}_{r,i} \uplus \mathcal{N}_{r,j}), \ldots, \mathcal{N}_{r,p}). \tag{12}$$

From what we said on the operations $\oplus$ and $\oslash$, after each elementary iteration of AVG-Merge, two invariants hold:

1. each peer l owns a sketch $\mathcal{D}_{r,l}$ which is a correct augmented sketch data structure for the portion of input stream $\mathcal{N}_{r,l}$;
2. $\biguplus_{l=1}^{p} \mathcal{N}_{r,l} = \mathcal{N}$.

These invariants remain true after each iteration of the main loop of AVG-Merge and, consequently, after each call to AVG-Merge, that is after each round of the P2PTFHH distributed protocol, when we derive from the vectors $\boldsymbol{\mathcal{D}}_r$ and $\boldsymbol{\mathcal{N}}_r$, the new vectors $\boldsymbol{\mathcal{D}}_{r+1}$ and $\boldsymbol{\mathcal{N}}_{r+1}$.

We can state that, for $r \to \infty$, the two vectors $\boldsymbol{\mathcal{D}}_r$ and $\boldsymbol{\mathcal{N}}_r$ converge respectively to:

$$\boldsymbol{\mathcal{D}}_\infty = \left(\mathcal{D}_{\text{avg}}, \mathcal{D}_{\text{avg}}, \ldots, \mathcal{D}_{\text{avg}}\right) \tag{13}$$

and

$$\boldsymbol{\mathcal{N}}_\infty = \left(\mathcal{N}_{\text{avg}}, \mathcal{N}_{\text{avg}}, \ldots, \mathcal{N}_{\text{avg}}\right), \tag{14}$$

where $\mathcal{N}_{\text{avg}} = \oslash_p(\mathcal{N})$ and $\mathcal{D}_{\text{avg}}$ is a correct FDCMSS sketch for $\mathcal{N}_{\text{avg}}$.

This means that all of the peers converge to a sketch of $\oslash_p(\mathcal{N})$, from which, for the properties of the operations $\oplus$ and $\oslash$, a correct sketch for $\mathcal{N}$ can be derived by computing $\oslash_{\frac{1}{p}}(\mathcal{D}_{\text{avg}})$, i.e. P2PTFHH converges. It is worth noting here that we actually need to know the number of peers p, which is not always the case.

However, p can be easily estimated by using the distributed averaging protocol, performed using the values initially distributed among the peers so that only one peer starts with its value equal to 1 while the other $p-1$ peers start with their values equal to 0. Denoting with $\tilde{q}_{r,l}$ the value held by peer l at round r, this quantity approaches the average value $1/p$ as the algorithm proceeds. Later, we shall provide in Section 5.3 a bound on the error for the estimation of p.

Thanks to the invariants discussed above, in order to prove the convergence of the local sketches to $\mathcal{D}_{\text{avg}}$, it is enough to verify that the local input streams implicitly induced by the algorithm converge to $\mathcal{N}_{\text{avg}}$.

We can represent each initial local input stream $\mathcal{N}_{0,l}$ for $l = 1, 2, \ldots, p$, as the frequencies' vector of the items in that stream, $\tilde{\boldsymbol{f}}_{0,l} = (\tilde{f}_{0,l,1}, \tilde{f}_{0,l,2}, \ldots, \tilde{f}_{0,l,m})$. Each value $\tilde{f}_{0,l,i}$ corresponds to the frequency that item i has in the initial local stream held by peer l. In this representation the operator $\oslash_p$ on a multiset simply multiplied the frequencies' vector corresponding to that multiset by the scalar $1/p$.

The implicit transformation that the local streams of the selected peers i and j undergo at each elementary iteration of AVG-Merge, is described by equation (12), which can be rewritten as:

$$\tilde{\boldsymbol{F}}'_r = (\tilde{\boldsymbol{f}}_{r,1}, \tilde{\boldsymbol{f}}_{r,2}, \ldots, \frac{1}{2}(\tilde{\boldsymbol{f}}_{r,i} + \tilde{\boldsymbol{f}}_{r,j}), \ldots, \frac{1}{2}(\tilde{\boldsymbol{f}}_{r,i} + \tilde{\boldsymbol{f}}_{r,j}), \ldots, \tilde{\boldsymbol{f}}_{r,p}), \tag{15}$$

where $\tilde{\boldsymbol{F}}_r$ is a matrix whose columns are the peers' vectors of frequencies after r rounds, i.e. each $\tilde{\boldsymbol{f}}_{r,l}$ is the frequencies' vector corresponding to the multiset $\mathcal{N}_{r,l}$. This matrix corresponds to the vector of multisets $\boldsymbol{\mathcal{N}}_r$ in equation (12).

Eventually, equation (15) can be recognised as the elementary step of the Jelasity's protocol applied component-wise to the frequencies' vectors of peers i and j. We already know that the Jelasity's averaging protocol converges to the average of the values initially owned by the peers. Thus, for $r \to \infty$, $\tilde{\boldsymbol{F}}_r$ converges to:

$$\tilde{\boldsymbol{F}}_\infty = (\boldsymbol{f}_{\text{avg}}, \boldsymbol{f}_{\text{avg}}, \ldots, \boldsymbol{f}_{\text{avg}}) \tag{16}$$

where $\boldsymbol{f}_{\text{avg}}$ is:

$$\boldsymbol{f}_{\text{avg}} = (\bar{f}_1, \bar{f}_2, \ldots, \bar{f}_m), \tag{17}$$

with $\bar{f}_i = \frac{1}{p}\sum_{l=1}^{p} \tilde{f}_{0,l,i}$, for $i = 1, 2, \ldots, m$ which is the representation as frequencies' vector of the multiset $\mathcal{N}_{\text{avg}}$ in equation (14), proving the convergence.

5.3. Gossip-based approximation

In our P2PTFHH algorithm, all of the peers participate to the averaging protocol by exchanging their internal state, i.e., by merging the local augmented sketches, $\mathcal{D}_{r,l}$, and averaging the local values $\tilde{q}_{r,l}$. We begin by deriving from Theorem 1 the following proposition that shall be used in the sequel.

Proposition 4. Let δ_g be a user-defined probability, $w_{r,l}$ the value held by peer l after r rounds of the averaging protocol, p the number of peers participating in the protocol, $\gamma = \mathbb{E}[2^{-\psi}] = 1/(2\sqrt{e})$ the convergence factor, $\bar{w}$ the mean of the initial vector of values $\boldsymbol{w}_0$, i.e. $\bar{w} = 1/p\sum_{l=1}^{p} w_{0,l}$ and $\bar{\epsilon} = \sqrt{\frac{\gamma^r}{\delta_g}}$ the error factor. Then, with probability greater than or equal to $1-\delta_g$ it holds that, for any peer l:

$$\left|w_{r,l} - \bar{w}\right| < \sqrt{(p-1)\sigma_0^2}\,\bar{\epsilon} < p\bar{w}\bar{\epsilon}. \tag{18}$$

Proof. From equation (5) it follows that:

$$\mathbb{E}[\sigma_r^2] = \mathbb{E}[2^{-\psi}]^r \sigma_0^2; \tag{19}$$

where σ_0^2 depends on the distribution of the initial numbers among the peers. By the Markov inequality, it holds that:

$$\mathbb{P}[\sigma_r^2 \geq \frac{\mathbb{E}[\sigma_r^2]}{\delta_g}] \leq \delta_g; \tag{20}$$

or

$$\mathbb{P}[\sigma_r^2 < \frac{\mathbb{E}[\sigma_r^2]}{\delta_g}] \geq 1 - \delta_g. \tag{21}$$

Considering the equations (4) and (19), it follows that:

$$\mathbb{P}[\sum_{l=1}^{p} (w_{r,l} - \bar{w})^2 < (p-1)\frac{\gamma^r \sigma_0^2}{\delta_g}] \geq 1 - \delta_g. \tag{22}$$

As a consequence, with probability at least $1 - \delta_g$:

$$max_{l\in[p]} (w_{r,l} - \bar{w})^2 \leq \sum_{l=1}^{p} (w_{r,l} - \bar{w})^2 < (p-1)\frac{\gamma^r \sigma_0^2}{\delta_g}, \tag{23}$$

which implies:

$$max_{l\in[p]} \left|w_{r,l} - \bar{w}\right| < \sqrt{(p-1)\sigma_0^2}\sqrt{\frac{\gamma^r}{\delta_g}}. \tag{24}$$

The initial distribution σ_0^2 of the elements over the peers typically is not known in advance, but the worst case happens when only one peer has the whole quantity $p\bar{w}$ and the other $p-1$ peers hold the value 0. In this case, it follows that $\sigma_0^2 \leq p\bar{w}^2$, and hence, with probability $1 - \delta_g$:

$$max_{l\in[p]} \left|w_{r,l} - \bar{w}\right| < \sqrt{(p-1)\sigma_0^2}\sqrt{\frac{\gamma^r}{\delta_g}} < p\bar{w}\bar{\epsilon}. \tag{25}$$

This proves the proposition. □

Equation (18) gives an upper bound on the error made by any peer estimating the value $\bar{w}$ after r rounds of the Jelasity's averaging algorithm. This bound is probabilistic and valid with probability greater than or equal to $1 - \delta_g$.

In order to estimate the number of peers, the averaging protocol is performed using the values initially distributed among the peers so that only one peer starts with its value equal to 1 while the other $p-1$ peers start with their values equal to 0. Denoting with $\tilde{q}_{r,l}$ the value held by peer l at round r, this quantity approaches the average value $1/p$ as the algorithm proceeds. We can bound the error related to the estimation of the number of peers. Considering Proposition 4 and its equation (18), we have that:

$$\left|\tilde{q}_{r,l} - \frac{1}{p}\right| < \bar{\epsilon}. \tag{26}$$

In fact, equation (26) can be easily derived from equation (18) by substituting $\bar{w}$ with the average value $1/p$. Since $\tilde{q}$ is an estimate of the inverse of the number of peer, we can write equation (26) introducing $\tilde{p} = 1/\tilde{q}$:

$$\frac{1}{p} - \bar{\epsilon} < \frac{1}{\tilde{p}_{r,l}} < \frac{1}{p} + \bar{\epsilon}. \tag{27}$$

Since the number of peers is a positive number, we can set the constraint $\bar{\epsilon} < 1/p$; under this constraint all of the members of the previous relation are positive, hence it holds that:

$$\frac{1}{1 + p\bar{\epsilon}} < \frac{\tilde{p}_{r,l}}{p} < \frac{1}{1 - p\bar{\epsilon}} \tag{28}$$

The problem with equation (28) is that the estimation error bounds depend on p, but we may not know p in advance. To overcome this problem, we introduce the value $p^* \geq p$, that is an estimate of the maximum number of peers we expect in the network, and we compute new bounds based on this value. Under the constraint $p^* \geq p$, we can be confident on the new computed bounds, though they may be weaker.

Let us set $\epsilon^* = p^*\bar{\epsilon}$. Given the constraint on $\bar{\epsilon}$, it holds that $0 < \epsilon^* < 1$, and, with probability $1 - \delta_g$, for any peer $l = 1, 2, \ldots, p$:

$$\frac{p}{1+\epsilon^*} < \tilde{p}_{r,l} < \frac{p}{1-\epsilon^*} \tag{29}$$

As discussed in Section 5.2, as the P2PTFHH algorithm proceeds, the augmented Sketch $\mathcal{D}_{r,l}$ held by a peer l after r rounds approaches $\mathcal{D}_{\text{avg}}$, which corresponds to the augmented sketch we would obtain had we applied FDCMSS to the stream $\mathcal{N}_{\text{avg}}$, which is, in turn, the stream corresponding to the average frequency vector that we would obtain applying the Jelasty averaging protocol component-wise to the local peers' frequency vectors.

Denoting with $\tilde{f}_{r,l,i}$ the decayed frequency of the item i in the stream $\mathcal{N}_{r,l}$ after r rounds, and considering the equation 18 of Proposition 4, it holds that:

$$\left|\tilde{f}_{r,l,i} - \frac{f_i}{p}\right| < f_i\bar{\epsilon}. \tag{30}$$

Equation (30) can be easily derived from equation (18) by substituting $\bar{w}$ with the average value f_i/p. Considering the expression of ϵ^*, it follows that, with probability $1 - \delta_g$, for any peer $l = 1, 2, \ldots, p$:

$$\frac{f_i}{p}(1-\epsilon^*) < \tilde{f}_{r,l,i} < \frac{f_i}{p}(1+\epsilon^*) \tag{31}$$

Finally, the total decayed count C shall be estimated too. Letting $\tilde{C}_{r,l}$ be the total decayed count estimate, it holds that $\tilde{C}_{r,l} = \sum_{i=1}^{d} \tilde{f}_{r,l,i}$. Thus, from equation (31) it follows that, with probability $1 - \delta_g$, for any peer $l = 1, 2, \ldots, p$:

$$\frac{C}{p}(1-\epsilon^*) < \tilde{C}_{r,l} < \frac{C}{p}(1+\epsilon^*). \tag{32}$$

Owing to the fact that the augmented sketch data structure is 1-norm equivalent to the classical Count-Min sketch (due to its properties), each peer can determine the total decayed count estimate $\tilde{C}_{r,l}$, simply by summing up the values of all of the cells in one row of its sketch, $\mathcal{D}_{r,l}$.

5.4. Correctness

We shall show here that, given an augmented sketch data structure $\mathcal{D}_{r,l}$ obtained by the peer l after r rounds of the P2PTFHH algorithm, we can select a set of items and their corresponding estimated frequencies, thus solving the Approximate Time–Faded Heavy Hitters Problem. We shall also determine the error bounds on frequencies' estimation and the relation among the number of the sketch's cells to be used by each peer and the number r of rounds to be executed in order to guarantee the false positives' tolerance requested by the user within a given probability of failure.

Theorem 5. Given an input stream $\mathcal{N}$ with total decayed count C, distributed among p peers, each one making use of an augmented sketch data structure of size $d \times w$, and given a threshold parameter $0 < \phi < 1$, after r rounds of P2PTFHH, the QUERY procedure executed on any peer returns a set H of items and their corresponding time–faded frequencies so that, with probability greater than $1 - \delta_n$, H includes all of the items in $\mathcal{N}$ with frequency $f > \phi C$, where $\delta_n = \delta_g + \left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d (1 - \delta_g)$ is a probability of failure, δ_g being the probability of failure pertaining to the gossip approximation.

Proof. We recap the query procedure of the P2PTFHH algorithm: all of the sketch's cells are inspected, and if the majority item candidate in the current cell has a decayed frequency greater than the selection criteria $\tau = \phi\tilde{C}_{r,l}\frac{1-\epsilon^*}{1+\epsilon^*}$, then a PointEstimate query is issued to compute the estimated frequency $\hat{f}$ of the heavy hitter candidate and finally if $\hat{f} > \tau$ the item is inserted in the set H.

Hence, to be selected a heavy hitter must:

- appear as a majority item candidate in at least one of the d cells in which it is mapped to by the hash functions;
- have estimated frequency $\hat{f}$ greater than τ.

We first prove that if an item is a heavy hitter (i.e., $f > \phi C$), then its estimated frequency is greater than τ with probability at least $1 - \delta_g$.

Owing to the FDCMSS properties it holds that:

$$\tilde{f}_{r,l,i} \leq \hat{f}_{r,l,i}, \tag{33}$$

and from the relations (29), (31), (32) and (33), we can derive the following:

$$\hat{f}_{r,l,i}\frac{p}{1-\epsilon^*} > \tilde{f}_{r,l,i}\frac{p}{1-\epsilon^*} > f_i > \phi C > \phi\tilde{C}_{r,l}\frac{p}{1+\epsilon^*}. \tag{34}$$

Hence it holds with probability at least $(1 - \delta_g)$ that

$$\hat{f}_{r,l,i} > \phi\tilde{C}_{r,l}\frac{1-\epsilon^*}{1+\epsilon^*}. \tag{35}$$

We now prove that a heavy hitter appears in at least one of the d cells in which it is mapped to by the hash functions with probability at least $\left[1 - \left(\frac{1}{2w\phi}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d\right](1 - \delta_g)$, and that the overall probability of P2PTFHH failing to retrieve all of the heavy hitters (or, alternatively, the probability of P2PTFHH reporting false negatives) is δ_n.

Let us take as a reference the state of a peer l after the rth round of the algorithm and let k be a frequent item for the global stream $\mathcal{N}$, i.e. $f_k > \phi C$. Let $\tilde{\mathcal{D}}_{r,l}[i][j]$, for $i \in \{1 \dots d\}$ and $j = h_i(k)$, be one of the d cells in which the item k is mapped to by the corresponding hash functions $h_i : [m] \to [w]$. Moreover, let $\tilde{f}^{(i,j)}_{min}$ be the minimum of the two counters available in the Space-Saving summary $\tilde{\mathcal{S}}_{i,j}$ monitoring the items falling in the cell $\tilde{\mathcal{D}}_{r,l}[i][j]$, and let $\hat{f}_{i,k}$ be the estimated decayed count of the item k related to the local stream $\tilde{\mathcal{N}}_{r,l}$ and determined through the Space-Saving summary $\tilde{\mathcal{S}}_{i,j}$.

The item k shall not appear as a majority item candidate in any of the d cells in which it is mapped to by the hash functions if and only if the following event happens:

$$\hat{f}_{i,k} \leq \tilde{f}^{(i,j)}_{min}, \forall i = 1, \dots, d. \tag{36}$$

Let $\tilde{S}_{i,j}$ be the total decayed count of the items falling in the cell $\tilde{D}_{r,l}[i][j]$. By equation (3), for the minimum of the two counters in the cell $\tilde{D}_{r,l}[i][j]$ it holds that: $\tilde{f}^{(i,j)}_{min} \leq \frac{\tilde{S}_{i,j}}{2}$.

We must now determine the probability that the event described in equation (36) occurs for a given row i of the sketch, i.e., $\Pr[\hat{f}_{i,k} \leq \tilde{f}^{(i,j)}_{min}]$, under the condition that $f_k > \phi C$.

Taking into account that $\tilde{f}^{(i,j)}_{min} \leq \frac{\tilde{S}_{i,j}}{2}$, it holds that:

$$\Pr[\hat{f}_{i,k} \leq \tilde{f}^{(i,j)}_{min}] < \Pr[\tilde{S}_{i,j} > 2\hat{f}_{i,k}], \tag{37}$$

and, as proved in [?], $\mathbb{E}[\tilde{S}_{i,j}] = \frac{\tilde{C}_{r,l}}{w}$. Thus, using the Markov inequality we have:

$$\Pr[\hat{f}_{i,k} \leq \tilde{f}^{(i,j)}_{min}] < \Pr[\tilde{S}_{i,j} > 2\hat{f}_{i,k}] \leq \frac{\mathbb{E}[\tilde{S}_{i,j}]}{2\hat{f}_{i,k}} = \frac{\tilde{C}_{r,l}}{2w\hat{f}_{i,k}}. \tag{38}$$

By applying equations (29)–(33) we have:

$$\Pr[\hat{f}_{i,k} \leq \tilde{f}_{min}^{(i,j)}] < \frac{\tilde{C}_{r,l}}{2w\hat{f}_{i,k}} < \frac{\tilde{C}_{r,l}}{2w\tilde{f}_{r,l,k}} < \frac{\tilde{C}_{r,l}}{2w\frac{f_k}{p}(1-\epsilon^*)} < \frac{\frac{C}{p}(1+\epsilon^*)}{2w\frac{f_k}{p}(1-\epsilon^*)}. \tag{39}$$

Finally, taking into account that k is a heavy hitter, i.e. $f_k > \phi C$, it holds that:

$$\Pr[\hat{f}_{i,k} \leq \tilde{f}_{min}^{(i,j)}] < \frac{1+\epsilon^*}{2\phi w(1-\epsilon^*)}. \tag{40}$$

We fail to identify a frequent item k when that item is not reported as a majority item candidate in any of the d cells $\tilde{D}_{r,l}[i][j]$ in which it falls. Thus, by equation (40), the probability of this event happening is:

$$\Pr\left[\bigwedge_{i=1}^{d}(\hat{f}_{i,k} \leq \tilde{f}_{min}^{(i,j)})\right] < \left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d. \tag{41}$$

Considering that the equations (29)–(33) are correct with a given probability $(1-\delta_g)$ (see Section 3.3), we can conclude that if an item is a heavy hitter, it is reported with probability greater than or equal to $\left[1-\left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d\right]\left(1-\delta_g\right)$, and that, consequently, the probability of failure of P2PTFHH retrieving all of the heavy hitters is:

$$\delta_n = \delta_g + \left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d(1-\delta_g). \tag{42}$$

$\square$

Theorem 6. Given an input stream $\mathcal{N}$ with total decayed count C, distributed among p peers, each one making use of an augmented sketch data structure of size $d \times w$, and given a threshold parameter $0 < \phi < 1$, after r rounds of P2PTFHH, the QUERY procedure executed on any peer returns a set H of items and their corresponding time–faded frequencies so that, with probability greater than or equal $1-\delta_p$, H does not include any item in $\mathcal{N}$ with frequency $f \leq (\phi-\epsilon)C$, where $\epsilon = \frac{4\epsilon^*\phi}{(1+\epsilon^*)^2} + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*}$ is a false positives tolerance, and $\delta_p = \delta_g + e^{-d}(1-\delta_g)$ is a probability of failure, δ_g being the probability of failure pertaining to the gossip approximation.

Proof. Owing to the FDCMSS properties, the estimated frequency of an item i is bounded by the following relation, with probability at least $1-e^{-d}$:

$$\hat{f}_{r,l,i} \leq \tilde{f}_{r,l,i} + \frac{e\tilde{C}_{r,l}}{2w}. \tag{43}$$

In order to compute the error, in terms of false positives' tolerance, that we commit with the selection criterion $\hat{f} > \tau$, we can use equations (29)–(32) and equation (43) and prove that if $\hat{f}_{r,l,i} > \phi\tilde{C}_{r,l}\frac{1-\epsilon^*}{1+\epsilon^*}$, then, with probability at least $(1-\delta_g)(1-e^{-d})$:

$$f_i > \left\{\phi - \left[\frac{4\epsilon^*\phi}{(1+\epsilon^*)^2} + \frac{e}{2w}\right]\right\}C. \tag{44}$$

Applying equation (43), if an item i is selected to be reported in the set H, then, with probability at least $(1-e^{-d})$:

$$\tilde{f}_{r,l,i} + \frac{e\tilde{C}_{r,l}}{2w} > \hat{f}_{r,l,i} > \phi\tilde{C}_{r,l}\frac{1-\epsilon^*}{1+\epsilon^*}, \tag{45}$$

which implies that

$$\frac{\tilde{f}_{r,l,i}}{\tilde{C}_{r,l}} + \frac{e}{2w} > \phi \frac{1-\epsilon^*}{1+\epsilon^*}. \tag{46}$$

Applying now equations (29)-(32), which are valid with probability at least $1-\delta_g$, we can state that, with probability at least $(1-\delta_g)(1-e^{-d})$ (i.e., with a probability of failure $\delta_p = \delta_g + e^{-d}(1-\delta_g)$), if the item i is reported in the output set H, then the following chain of implications holds:

$$\begin{aligned}
&\frac{f_i(1+\epsilon^*)}{C(1-\epsilon^*)} + \frac{e}{2w} > \phi \frac{1-\epsilon^*}{1+\epsilon^*} \implies \\
&f_i > \phi C \left(\frac{1-\epsilon^*}{1+\epsilon^*}\right)^2 - \frac{eC}{2w}\frac{1-\epsilon^*}{1+\epsilon^*} \implies \\
&f_i > \left\{\phi - \left[1 - \left(\frac{1-\epsilon^*}{1+\epsilon^*}\right)^2\right]\phi + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*}\right\} C \implies \\
&f_i > \left\{\phi - \left[\frac{4\epsilon^*\phi}{(1+\epsilon^*)^2} + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*}\right]\right\} C.
\end{aligned} \tag{47}$$

□

Theorems 5 and 6 show that the set of candidate heavy hitters produced by P2PTFHH satisfies the requirements of the Approximate Time Faded Heavy Hitters problem (Problem 1). Thus, we can set the parameters of the algorithm, i.e. the dimensions of the sketches and/or the number of rounds to be executed, so that the absence of false negatives and the tolerance on false positives are guaranteed with certain probabilities of failure that in turn depend on the algorithm's parameters and can be kept small.

Thus, the overall probability of failure of P2PTFHH is driven by δ_p and we can avoid dealing with the two distinct probabilities of failure, δ_n and δ_p.

We know from Problem 1 that the desired value for the tolerance ϵ must be such that $\epsilon < \phi$; therefore, starting from the expression of ϵ from Theorem 6, the following chain of implications holds:

$$\begin{aligned}
&\frac{4\epsilon^*\phi}{(1+\epsilon^*)^2} + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*} < \phi \implies \\
&\left[\frac{4\epsilon^*}{(1+\epsilon^*)^2} - 1\right]\phi + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*} < 0 \implies \\
&\frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*} - \left(\frac{1-\epsilon^*}{1+\epsilon^*}\right)^2 \phi < 0 \implies \\
&\frac{e}{2w} - \phi\frac{1-\epsilon^*}{1+\epsilon^*} < 0 \implies \\
&w > \frac{e}{2\phi}\frac{1+\epsilon^*}{1-\epsilon^*}.
\end{aligned} \tag{48}$$

Under the condition above, we can see that $\delta_n < \delta_p$. In fact:

$$\begin{aligned}
w > \frac{e}{2\phi}\frac{1+\epsilon^*}{1-\epsilon^*} &\implies \left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right) < \frac{1}{e} \\
&\implies \delta_g + \left(\frac{1}{2\phi w}\frac{1+\epsilon^*}{1-\epsilon^*}\right)^d (1-\delta_g) < \delta_g + \left(\frac{1}{e}\right)^d (1-\delta_g) \\
&\implies \delta_n < \delta_p
\end{aligned} \tag{49}$$

We are now ready to state the following theorem, which follows straight from previous results and reasoning, and subsumes Theorems 5 and 6.

Theorem 7. Given an input stream $\mathcal{N}$ with total decayed count C, distributed among p peers, each one making use of an augmented sketch data structure of size $d \times w$, and given a threshold parameter $0 < \phi < 1$, after r rounds of P2PTFHH, the QUERY procedure executed on any peer returns a set H of items and their corresponding time–faded frequencies so that, with probability greater than $1 - \delta$:

1. H includes all of the true time–faded heavy hitters ($f > \phi C$);
2. H does not include any item in $\mathcal{N}$ with frequency $f \leq (\phi - \epsilon)C$;

where $\epsilon = \frac{4\epsilon^* \phi}{(1+\epsilon^*)^2} + \frac{e}{2w}\frac{1-\epsilon^*}{1+\epsilon^*}$ is a false positives tolerance, and $\delta = \delta_g + e^{-d}(1 - \delta_g)$ is a probability of failure, δ_g being the probability of failure pertaining to the averaging gossip protocol.

5.5. Error bounds on the estimated frequency

The estimated frequencies provided by $\mathcal{D}_{r,l}$ refer to average frequencies. Thus, in order to obtain an estimate of the global frequency f_i of an item i, we need to multiply $\hat{f}_{r,l,i}$ by $\tilde{p}_{r,l}$. From equations. (29)–(33) and (43) we can compute the corresponding error bounds. The following theorem holds.

Theorem 8. Given an input stream $\mathcal{N}$ with total decayed count C, distributed among p peers, each one making use of an augmented sketch data structure of size $d \times w$, and given a threshold parameter $0 < \phi < 1$, after r rounds of P2PTFHH, any peer can report a time–faded estimated frequency $f^s_{r,l,i}$ of an item $i \in [m]$ so that, with probability $1 - \delta$:

$$\frac{1-\epsilon^*}{1+\epsilon^*} f_i < f^s_{r,l,i} < \frac{1+\epsilon^*}{1-\epsilon^*}\left(f_i + \frac{eC}{2w}\right), \tag{50}$$

where $\delta = \delta_g + e^{-d}(1-\delta_g)$ is a probability of failure, δ_g being the probability of failure pertaining to the gossip approximation.

Proof. From equation (29), with probability greater than $1 - \delta_g$:

$$\frac{1}{1+\epsilon^*} < \frac{\tilde{p}_{r,l}}{p} < \frac{1}{1-\epsilon^*}, \tag{51}$$

and from equation (31) and equation (32), with probability greater than $1 - \delta_g$:

$$\begin{aligned} f_i \frac{\tilde{p}_{r,l}}{p}(1-\epsilon^*) &< \tilde{f}_{r,l,i}\tilde{p}_{r,l} < f_i \frac{\tilde{p}_{r,l}}{p}(1+\epsilon^*), \\ C\frac{\tilde{p}_{r,l}}{p}(1-\epsilon^*) &< \tilde{C}_{r,l}\tilde{p}_{r,l} < C\frac{\tilde{p}_{r,l}}{p}(1+\epsilon^*). \end{aligned} \tag{52}$$

Now, starting from equations (33) and (43), and taking into account equations (51) and (52), the following chain of implications holds with probability greater than $(1 - \delta_g)(1 - e^{-d})$:

$$\begin{aligned} \tilde{f}_{r,l,i}\tilde{p}_{r,l} \leq \hat{f}_{r,l,i}\tilde{p}_{r,l} &\leq \tilde{f}_{r,l,i}\tilde{p}_{r,l} + \frac{e\tilde{C}_{r,l}}{2w}\tilde{p}_{r,l} \implies \\ f_i\frac{\tilde{p}_{r,l}}{p}(1-\epsilon^*) < \hat{f}_{r,l,i}\tilde{p}_{r,l} &< \left(f_i + \frac{eC}{2w}\right)\frac{\tilde{p}_{r,l}}{p}(1+\epsilon^*) \implies \\ \frac{1-\epsilon^*}{1+\epsilon^*} f_i < \hat{f}_{r,l,i}\tilde{p}_{r,l} &< \frac{1+\epsilon^*}{1-\epsilon^*}\left(f_i + \frac{eC}{2w}\right). \end{aligned} \tag{53}$$

Eventually, setting $f^s_{r,l,i} = \hat{f}_{r,l,i}\tilde{p}_{r,l}$, the bounds stated in equation (50) follow. □

5.6. Practical considerations

We conclude this Section discussing how to select proper values for the parameters d, w and R, which represent respectively the number of rows and columns of the augmented sketch data structure and the minimum number of rounds required to solve the Approximate Time–Faded Heavy Hitters Problem with a given threshold ϕ, tolerance ϵ and probability of failure δ. Theorem 7 proves the correctness of the P2PTFHH algorithm, also providing the relation which ties the value of ϵ to the parameters of the algorithm.

Let us consider as fixed the values of p^* and δ_g. Then, in order to guarantee that the desired false positives tolerance ϵ is not exceeded, the user can set the number of rounds R and/or the number of sketch columns w. By fixing the value of ϵ, the relation between w and R is given by equation (54).

$$w = \frac{e(1-\epsilon^{*2})}{2\epsilon\,(1+\epsilon^*)^2 - 8\phi\epsilon^*} = \frac{e(1-p^{*2}\frac{\gamma^R}{\delta_g})}{2\epsilon\left(1+p^*\sqrt{\frac{\gamma^R}{\delta_g}}\right)^2 - 8\phi p^*\sqrt{\frac{\gamma^R}{\delta_g}}} \tag{54}$$

Among all of the possible values for R and w, the user could follow a strategy oriented to maintain the number of rounds (hence the running time) as fewer as possible and to choose w accordingly or vice-versa to maintain the number of counters (hence the space) as lower as possible and to choose R accordingly. Let us now discuss both strategies.

With the first strategy, which can be called time-dominant, the user is interested to choosing the R and w parameters which guarantee a given ϵ such that R is minimum. Equation (54) reveals that R is a monotone decreasing function with regard to w, hence the minimum value for R is obtained when w tends to infinity; moreover, since $w > 0$ the minimum value for R can be computed by imposing the following constraint:

$$2\epsilon\left(p^*\sqrt{\frac{\gamma^R}{\delta_g}}+1\right)^2 - 8\phi p^*\sqrt{\frac{\gamma^R}{\delta_g}} > 0, \tag{55}$$

from which it follows that

$$R > \frac{\log\delta_g + 2\log\left(\frac{2\phi-\epsilon-2\sqrt{\phi^2-\epsilon\phi}}{\epsilon p^*}\right)}{\log\gamma}. \tag{56}$$

Since R is an integer, the minimum value of R is given by:

$$R_{min} = \left\lfloor \frac{\log\delta_g + 2\log\left(\frac{2\phi-\epsilon-2\sqrt{\phi^2-\epsilon\phi}}{\epsilon p^*}\right)}{\log\gamma} \right\rfloor + 1. \tag{57}$$

Substituting the value of R_{min} provided by equation (57) into equation (54) for R, it is possible to obtain the value for w.

With the second strategy, which can be called space-dominant, the user wants to keep the memory footprint as low as possible. Equation (54) reveals that w is a monotone decreasing function with regard to R, hence the minimum value for w is obtained when R tends to infinity. Evaluating equation (54) for $R \to \infty$ it holds that the minimum value for w is given by:

$$w > \frac{e}{2\epsilon}. \tag{58}$$

Considering that w is an integer value,

$$w_{min} = \left\lfloor \frac{e}{2\epsilon} \right\rfloor + 1. \tag{59}$$

Solving equation (54) by ϵ^* and using equation (59) it holds that:

$$\epsilon^* = \frac{2w_{min}(2\phi - \epsilon) - \sqrt{16\phi w_{min}^2(\phi - \epsilon) + e^2}}{e + 2\epsilon w_{min}}. \tag{60}$$

Since $\epsilon^* = p^* \sqrt{\frac{\gamma^R}{\delta_g}}$, it holds that:

$$R = \frac{1}{\log \gamma}\left(2 \log \epsilon^* - 2 \log p^* + \log \delta_g\right). \tag{61}$$

And, since R is an integer,

$$R = \left\lfloor \frac{1}{\log \gamma}\left(2 \log \epsilon^* - 2 \log p^* + \log \delta_g\right) \right\rfloor + 1. \tag{62}$$

Finally, we discuss how the parameters of the algorithm influence the probability of failure δ. P2PTFHH is a probabilistic algorithm in which two sources of randomness occur: the first is due to the way the information is spread over the peers at each round of the gossip protocol and is represented by the probability δ_g; the second is related to the mapping of an item to a cell of the augmented sketch data structure through the corresponding hash function. The overall probability of failure of P2PTFHH in solving the Problem 1 with the desired guarantees can be controlled and made as small as desired, by acting on the parameter d, i.e., the number of rows of the sketch, and the value of δ_g, that in turn depends on the value of R.

Theorem 7 states that the Approximated Time–Faded Heavy Hitters Problem can be solved without false negative items and with a bounded number of false positive items with probability of failure less than $\delta = \delta_g + e^{-d}(1 - \delta_g)$. Thus, given a desired value of δ, we have to choose d such that:

$$d \geq \log \frac{1 - \delta_g}{\delta - \delta_g}. \tag{63}$$

6. Experimental results

We implemented a simulator in C++ using the igraph library [?] in order to test the performance of our algorithm. The simulator has been compiled using the GNU C++ compiler g++ 4.8.5 on CentOS Linux 7. The tests have been performed on a machine equipped with two hexa-core Intel Xeon-E5 2620 CPUs at 2.0 GHz and 64 GB of main memory. The source code of the simulator is freely available for inspection and for reproducibility of the results contacting the authors by email.

Objective of the experiments was to provide evidence of P2PTFHH's behaviour under various conditions, through the use of some useful and commonly referred metrics: Recall, Precision, and frequency estimation error. Recall is defined as the fraction of heavy hitters retrieved by an algorithm over the total number of heavy hitters. Precision is the fraction of heavy hitters retrieved over the total number of items reported as heavy hitters candidates. The frequency estimation error that we measure and report is the Average Relative Error, ARE, committed in the frequency estimation of items outputted by the algorithm as frequent candidates. The Relative Error is defined as usual as $\frac{|f^S - f|}{f}$, where f^S is the frequency reported for an item and f is its true frequency.

We provide results for both synthetically generated data and real datasets. Regarding synthetic data, for every experiment, a global input stream of items has been generated following a Zipfian distribution (items are 32 bits unsigned integers) and each peer has been assigned a distinct part of that global stream, thus simulating the scenario in which each peer processes, independently of the other peers, its own local sub-stream. The peers collaboratively discover the heavy hitters of the union of their sub-streams.

The experiments have been repeated 10 times setting each time a different seed for the pseudo-random number generator used for creating the input data. For each execution, we collected the peers' statistics discussed above. Then, for each peer we determined the average value of those statistics over the ten executions. At last, we computed the mean and confidence interval of each statistics over all of the peers and plotted these values.

Table 2: Parameter values for the experiments on synthetic data

ρ	ϕ	p ($\times 10^3$)	sw ($\times 10^3$)	r	fan-out
{0.9, 1.1, 1.3, 1.5}	{0.01, 0.02, 0.03, 0.04}	{1, 5, 10, 15}	{1.5, 2.5, 3.5, 4.5}	{21, 24, 27}	{1, 3}
1,2	0.02	5	2.5	24	1

Table 3: Parameter values for the experiments on real datasets

ϕ	p ($\times 10^3$)	sw ($\times 10^3$)	r	fan-out
{0.001, 0.002, 0.003, 0.004}	{0.5, 1, 2, 5}	{1.5, 2.5, 3.5, 4.5}	{18, 20, 22, 24}	{1, 3}
0.002	2	2.5	24	1

We fixed the size of the global stream at 100 millions of items, and varied each one of the other parameters: the skew of the Zipfian distribution, ρ, the number of peers, p, the heavy hitters threshold, ϕ, the width of the sketch used by each peer, sw, (while the depth is fixed to 4) , and the fan-out, setting non varying parameters to default values. Every experiment has been carried out by generating random P2P network topologies through both the Barabasi-Albert and the Erdos-Renyi random graphs models. Table 2 reports the sets of values (first row) and the default values (second row) used for each parameter. All of the plots representing the evolution of Recall and Precision are on linear-linear scale, whilst the plots that show the behaviour of the relative error are on linear-logarithmic scale.

Regarding real datasets, we show the results for two different datasets. The first real dataset has been derived by a publicly available traffic trace [1] [?]. The trace includes about 62 millions of packets. By using the Wireshark tool, we extracted the first 20 millions of packets and created the dataset. In order to carry out our experiments, when processing a dataset, a flow (identified by a tuple consisting of source IP address, source port, destination IP address, destination port and protocol) has been mapped to an item. We shall refer to this dataset as CAIDA.

The second real dataset, called WEBDOCS, is a publicly available spidered collection of web html documents. The whole collection contains about 1.7 millions documents, mainly written in English, and its size is about 5 GB. The resulting dataset, after preliminary filtering and pre-processing, has a size of about 1.48 GB. We extracted the first 20 millions items.

Table 3 reports the sets of values (first row) and the default values (second row) used for each parameter in the experiments on real datasets.

We begin reporting the experimental results obtained respectively by using a static network topology in which the set of agents does not vary (no churn).

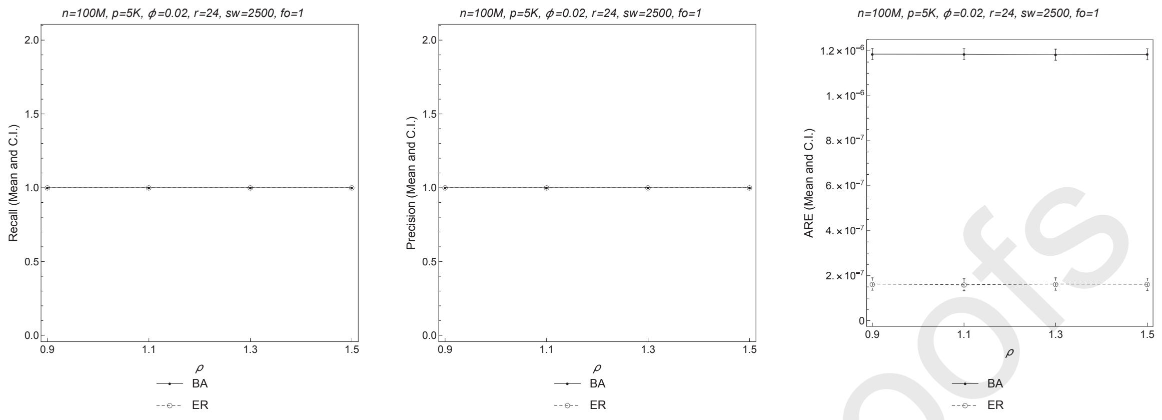
6.1. Static network topology: synthetic data

Figures 1-5 report the results of the experiments carried out without taking into account the peer's churning. The Recall, Precision and the Average Relative Error measured when varying the skewness of the input distribution are respectively depicted by Fig. 1a, Fig. 1b and Fig. 1c. Recall and Precision are always 100%, i.e., the algorithm is very robust with regard to skewness' variations in the input. Moreover, we observe very low Average Relative Errors on frequency estimation, both in the Barabasi-Albert graph (the worst) and the Erdos-Renyi graph (the better).

Figure 2 shows how P2PTFHH behaves with regard to variations of the threshold ϕ. Even in this setting Recall (Fig. 2a) is always 100%. On the other hand, Precision (Fig. 2b) presents a value slightly below 100% only when $\phi = 0.01$. Average Relative Errors (Fig. 2c) confirm what already told about the skewness' plots.

Figure 3 plots the results of the experiments in which we varied the number of peers, executing a scalability analysis. Recall and Precision stay at the maximum value, whilst the Average Relative Error

[1] https://data.caida.org/datasets/passive-2018/equinix-nyc/20180315-130000.UTC/equinix-nyc.dirB.20180315-125910.UTC.anon.pcap.gz



(a) Recall (b) Precision (c) Average Relative Error

Figure 1: Recall, Precision and Average Relative Error (mean and confidence interval) varying the skewness of the input distribution, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

(Fig. 3c) slightly increases when the number of peers grows. Figure 3c also shows that the error decreases when the fan-out increases from 1 to 3. This behaviour is expected, since with a greater fan-out the number of messages exchanged in a round increases and the information spreads quickly. If the number of rounds executed and the dimension of the sketch used by the peers is not adequate to the number of peers in the network, the metrics tend to get worse.

The plots related to the experiments in which we varied the width *sw* of peers' sketch (Figure 4) do not present a particular behaviour in the interval of values tested, showing that in this case the sketch's dimension used was always enough with regard to the number of rounds executed in order to guarantee a good accuracy.

A major sensitivity is exhibited by the algorithm when the number of rounds executed is varied, as shown in Figure 5. We note that the Precision grows and the Average Relative Error decreases as the number of rounds increases. This behaviour is expected, given the theoretical analysis.

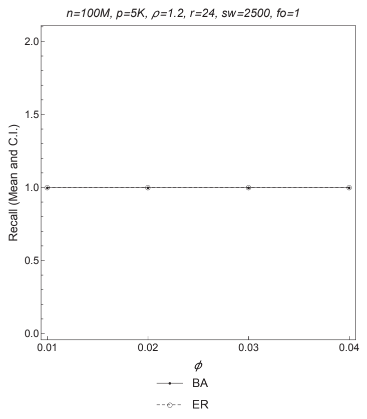
Overall the experiments show that our algorithm exhibits very good performance in terms of Recall, Precision, and Average Relative Error of the frequency estimation when the guidance of the theoretical analysis is taken into account in determining the size of the sketch used by peers and the number of rounds to be executed. Furthermore, the algorithm proves to be very robust to variations in the skewness of the input dataset and the heavy hitters threshold.

6.2. Static network topology: real datasets

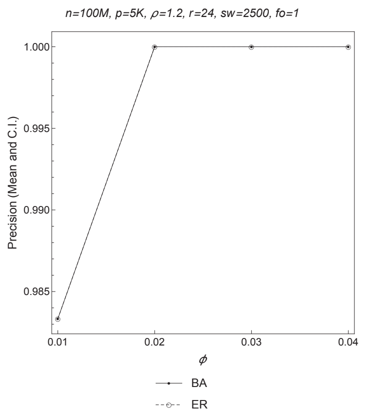
Here, we present the results of the experiments carried out on real datasets without taking into account churning.

Regarding the CAIDA dataset, we do not report the recall obtained when varying the number of peers, the threshold ϕ, the number of rounds executed and the amount of space used, owing to the fact that in all of the cases, the recall is always 100%. Figures 6 and 7 depict respectively the precision and ARE obtained for both a Barabasi-Albert and an Erdos-Renyi type of network graph when varying the number of peers, the threshold ϕ, the number of rounds executed and the amount of space used. As shown, the precision is almost always 100% or near 100%, except in a few cases, in which the value attained is anyway 80%. Only when the number of rounds is quite small, the precision is low, but this is an expected obvious result. ARE values are very close to zero in all of the conditions under test.

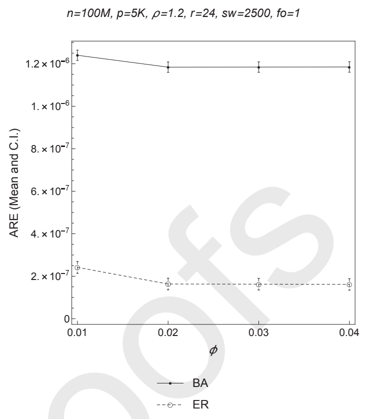
For the WEBDOCS dataset, as shown by Figures 8 and 9 we can draw the same conclusions, considering that the precision is always greater than or equal to about 88%, except when rounds are too few, and that the ARE values are even closer to zero than those related to the CAIDA dataset.



(a) Recall



(b) Precision



(c) Average Relative Error

Figure 2: Recall, Precision and Average Relative Error (mean and confidence interval) varying the heavy hitters threshold ϕ, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

6.3. Time-varying network topology (churn): synthetic data

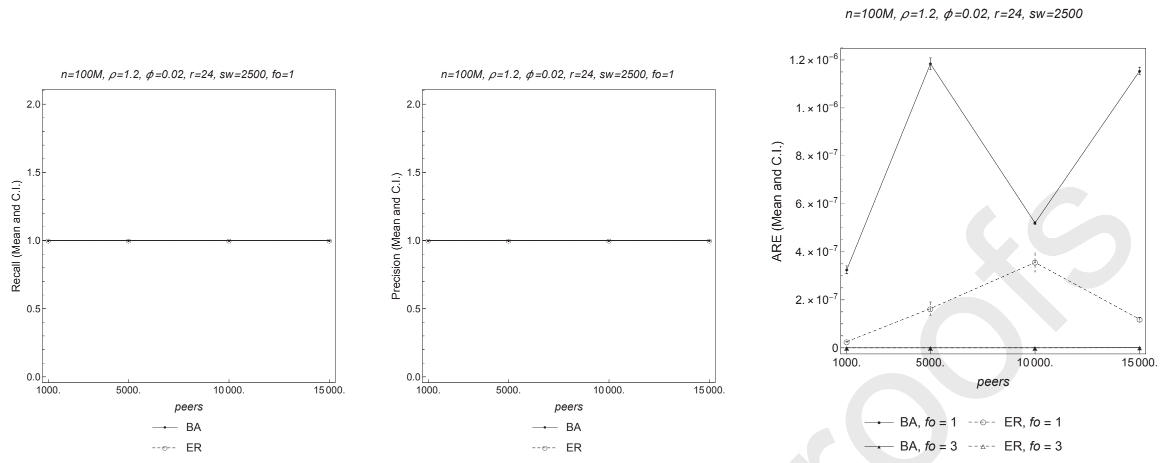
We now report the experimental results obtained respectively by using a time-varying network topology in which the set of agents is subject to churn. The performance of our P2PTFHH algorithm has been tested also in more realistic contexts with the adoption of two churning models, the fail-stop model and the Yao model, proposed by Yao et al. [?]. In the fail-stop model, a peer could leave the network with a given failure probability and the failed peers can not join the network anymore. In the Yao model, the peers randomly join and leave the network. For each peer i, a random average lifetime duration l_i is generated from a Shifted Pareto distribution with parameters $\alpha = 3$, $\beta = 1$ and $\mu = 1.01$. In the same way, a random average offline duration d_i is generated from a Shifted Pareto distribution using the same α and μ values and $\beta = 2$. We recall here that if $X \sim \text{Pareto(II)}(\mu, \beta, \alpha)$, i.e., X is a random variable with a Pareto Type II distribution (also named Shifted Pareto), then its cumulative distribution function is $F_X(x) = 1 - \left(1 + \frac{x-\mu}{\beta}\right)^{-\alpha}$.

The values l_i and d_i are used to set, for each peer i, two distributions F_i and G_i.The distributions G_i are Shifted Pareto distributions with $\beta = 3$ and $\alpha = 2d_i$. Whilst, the distributions F_i can be both Pareto distributions with $\beta = 2$, $\alpha = 2l_i$, or exponential distributions with $\lambda = 1/l_i$. Whenever the state of a peer changes, a duration value is drawn from one of the distributions, F_i or G_i, based on the type of duration values (lifetime or off-time) which must be generated. We run our experiments with both variants, i.e., Pareto and Exponential lifetimes.

When using the fail-stop model, we run our algorithm with the default parameter values of Table 2 and varying the failure probability through the values: $0.0, 0.01, 0.05, 0.1$.

As shown in Figures 10a and 10b, the recall and precision metrics are not affected by the introduction of peer failures up to a failure probability equal to 0.1. However, as expected, the average relative error on frequency estimations gets worse going from about 10^{-7} in case of no churn to about 10^{-1} when the failure probability is 0.1, as depicted by Figure 10c.

Using the Yao model of churning, we run our algorithm with the default values of Table 2 and the churning parameters already discussed, varying the number of peers available to participate to the gossip protocol (scalability analysis) and the number of rounds executed. Also in these cases, recall and precision are not affected by the introduction of churning. In fact, we obtained for recall and precision varying the number of peers and the number of rounds the same plots shown in Figures 3a, 3b and 5a and 5b. For this reason we do not report these plots again. On the other hand, the average relative error is affected by the churning, as expected: Figures 11a and 11b are related respectively to the ARE measured varying the number of peers and the number of rounds with Pareto distributions for lifetimes, whilst Figures 11c and 11d refer to the ARE measured when using Exponential distributions for lifetimes.



(a) Recall (b) Precision (c) Average Relative Error

Figure 3: Recall, Precision and Average Relative Error (mean and confidence interval) varying the number of peers participating in the computation, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph, and setting a fan-out equal to 1 and 3 in case of the ARE plot.

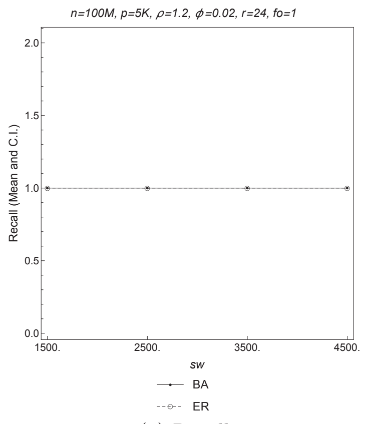
6.4. Time-varying network topology (churn): real datasets

Figures 12 and 13 depict the result obtained varying the failure probability in a fail-stop model of churning, for both a Barabasi-Albert and an Erdos-Renyi type of network graph, respectively for the CAIDA and the WEBDOCS datasets. As expected, both recall and precision decrease when the failure probability increases; in particular, the WEBDOCS datasets exhibits a better behaviour for heavy hitter detection. Regarding the ARE, we can conclude that frequency estimation is affected by a large error.

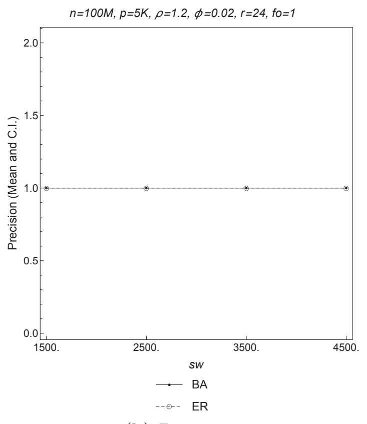
Regarding the Yao model of churning, the behaviour of the CAIDA dataset is shown in Figures 14-16. The recall is extremely high, attaining values equal to 100% or near 100%. Moreover, the precision exhibits the same behaviour, and, as expected, it increases when the number of rounds executed increases. As a consequence, heavy hitter detection is excellent even in this case. On the contrary, the ARE values are large enough to adversely affect the frequency estimation. Regarding the WEBDOCS dataset, we can infer the same conclusions by inspecting the results provided in Figures 17-19.

7. Conclusions

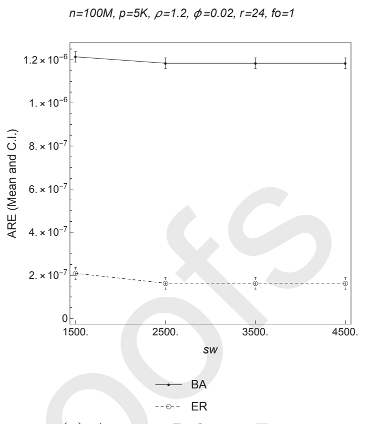
We have presented the P2PTFHH (Peer–to–Peer Time–Faded Heavy Hitters) distributed algorithm. Building on the distributed averaging protocol, we have designed a gossip–based version of our sequential FDCMSS (Forward Decay Count-Min Space-Saving) algorithm for distributed mining of time–faded heavy hitters. To the best of our knowledge, this is the first distributed protocol designed specifically for the problem of mining time–faded heavy hitters on unstructured P2P networks, a data mining task with wide applicability. For instance, applications arise in the context of sensor data mining, for business decision support, analysis of web query logs, network measurement, monitoring and traffic analysis. In these applications the use of the time–fading model discounts the effect of old data, by putting heavier weights on recent batches of streaming data than older batches. We have formally proved the algorithm's correctness and its error bounds on frequency estimation. The extensive experimental results, executed on both synthetic and real datasets, also have proved that P2PTFHH is extremely accurate and fast, allowing near real time processing of large datasets.



(a) Recall



(b) Precision

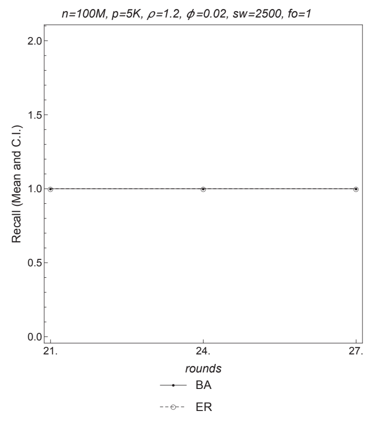


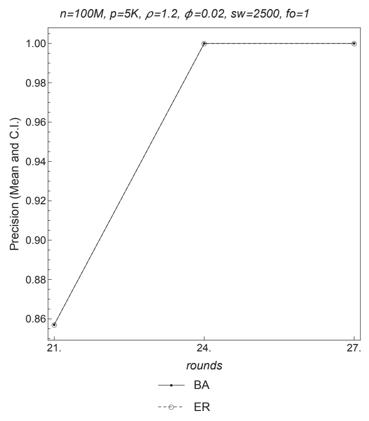
(c) Average Relative Error

Figure 4: Recall, Precision and Average Relative Error (mean and confidence interval) varying the number of Space-Saving counters used by each peer, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

Acknowledgment

We are indebted to the unknown referees for enlightening observations, which helped us to improve the paper.


(a) Recall


(b) Precision

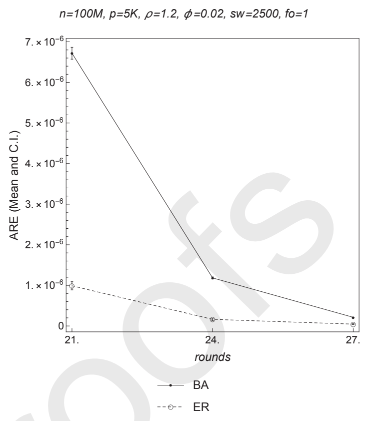
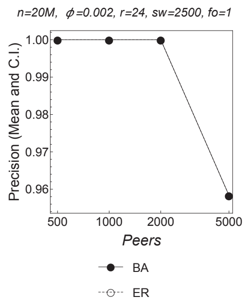
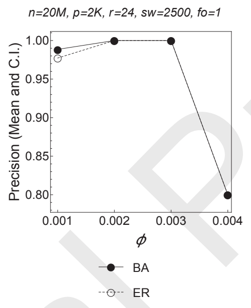
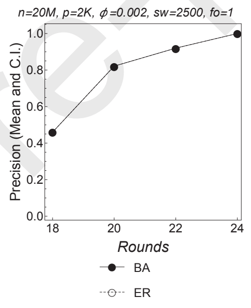

(c) Average Relative Error

Figure 5: Recall, Precision and Average Relative Error (mean and confidence interval) varying the number of rounds executed, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.


(a)


(b)


(c)

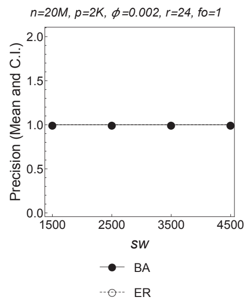
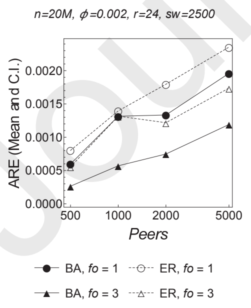
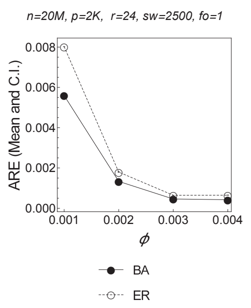
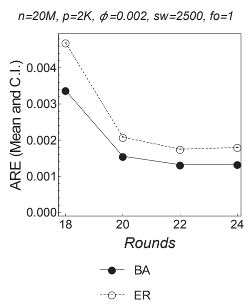

(d)

Figure 6: Dataset CAIDA: Precision (mean and confidence interval) varying (a) the number of peers, (b) the threshold ϕ, (c) the number of rounds executed and (d) the space used, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.


(a)


(b)


(c)

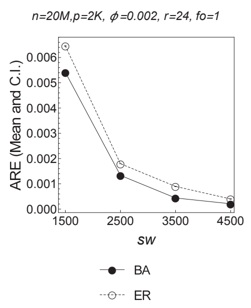
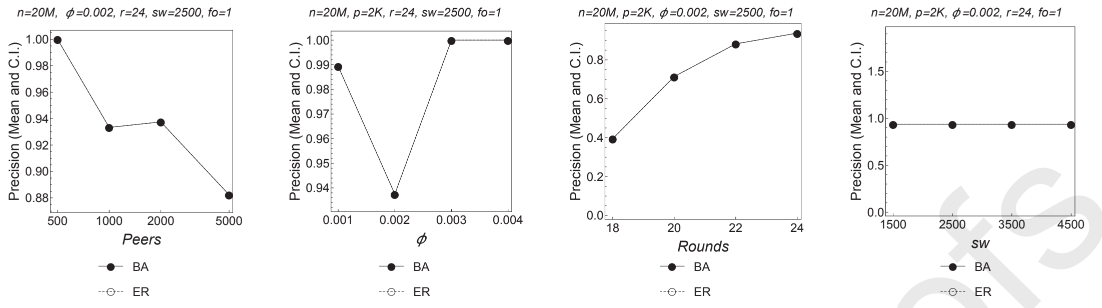
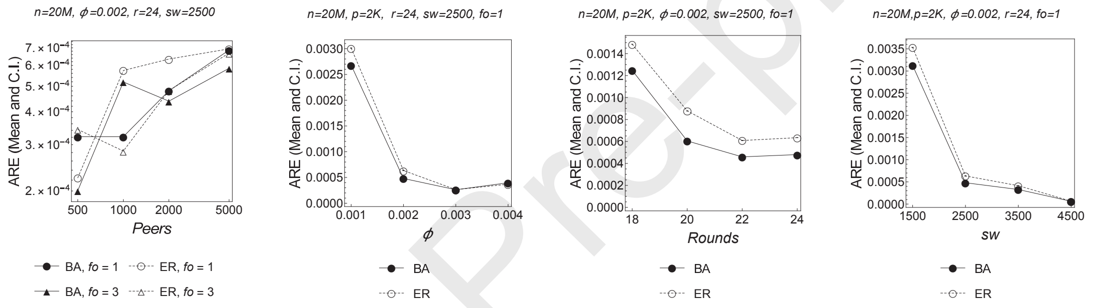

(d)

Figure 7: Dataset CAIDA: Average Relative Error (mean and confidence interval) varying (a) the number of peers, (b) the threshold ϕ, (c) the number of rounds executed and (d) the space used, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.



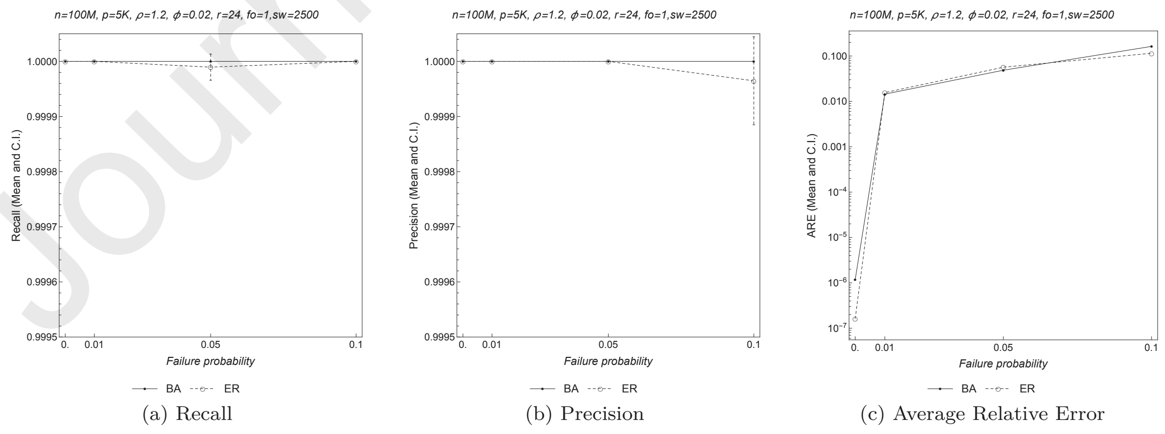
(a) (b) (c) (d)

Figure 8: Dataset WEBDOCS: Precision (mean and confidence interval) varying (a) the number of peers, (b) the threshold ϕ, (c) the number of rounds executed and (d) the space used, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.



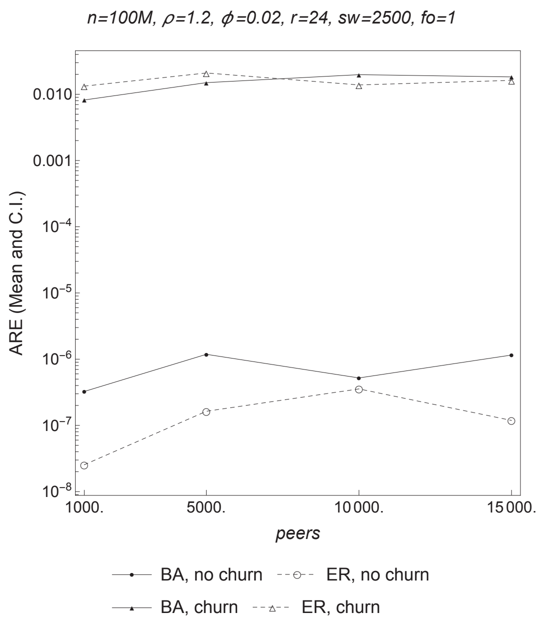
(a) (b) (c) (d)

Figure 9: Dataset WEBDOCS: Average Relative Error (mean and confidence interval) varying (a) the number of peers, (b) the threshold ϕ, (c) the number of rounds executed and (d) the space used, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

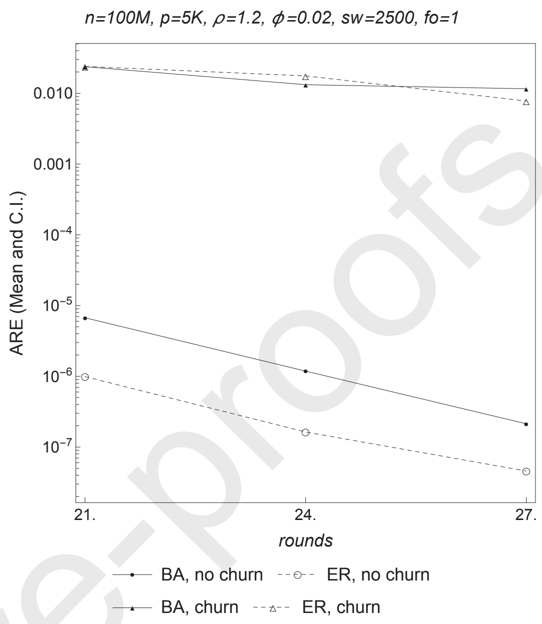


(a) Recall (b) Precision (c) Average Relative Error

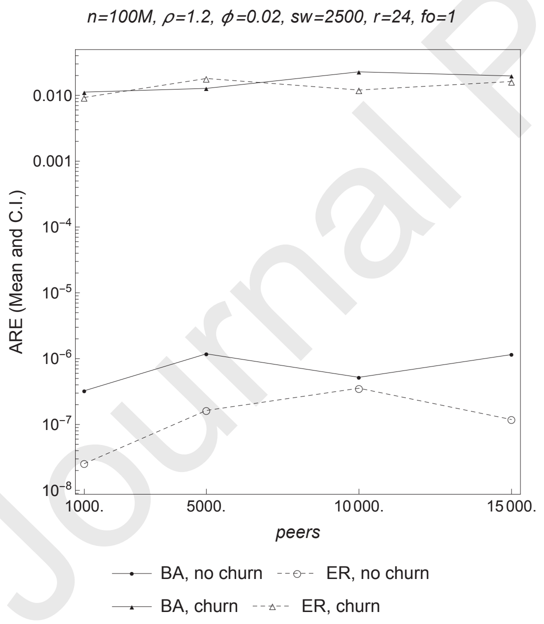
Figure 10: Recall, Precision and Average Relative Error (mean and confidence interval) varying the failure probability in a fail-stop model of churning, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.



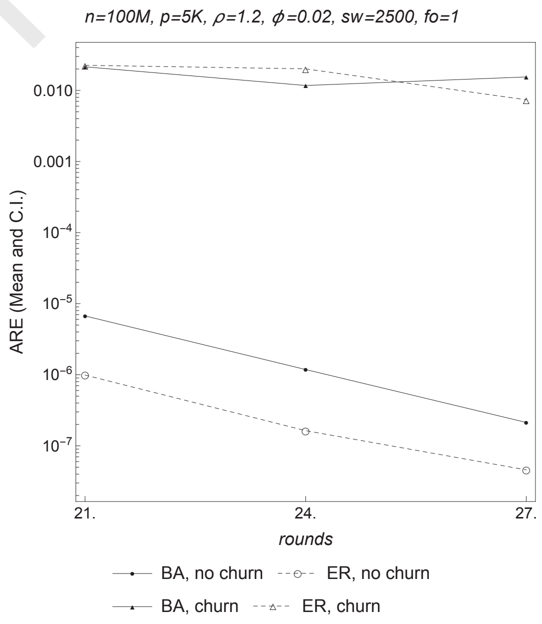
(a) Average Relative Error



(b) Average Relative Error

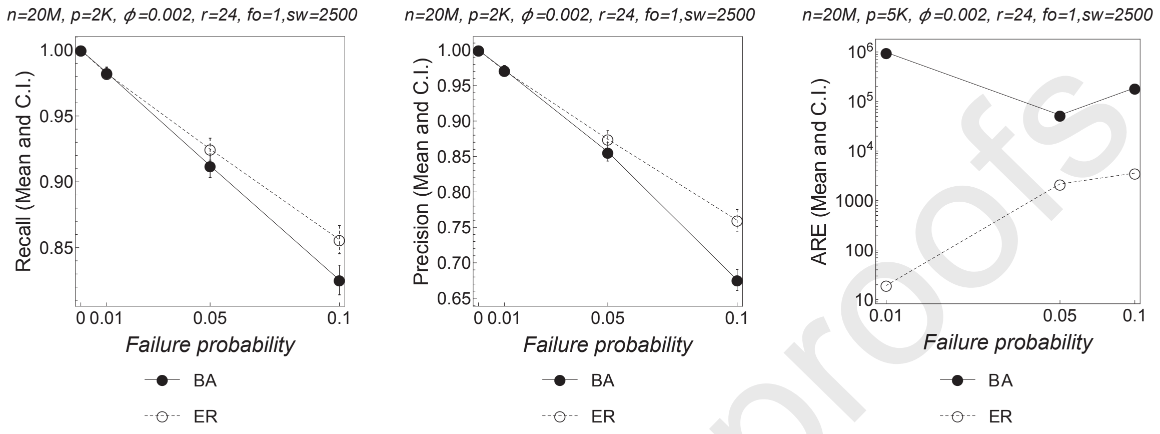


(c) Average Relative Error



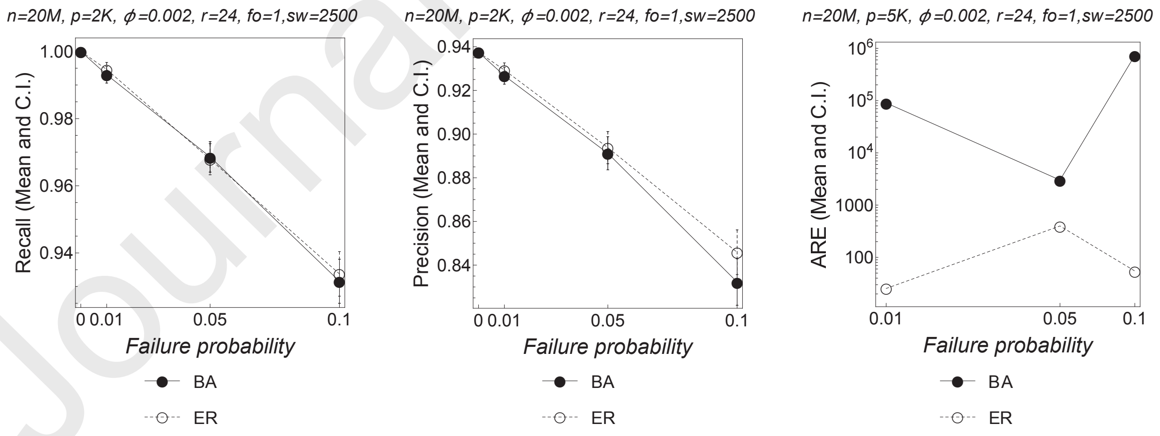
(d) Average Relative Error

Figure 11: Average Relative Error (mean and confidence interval) varying the number of peers and rounds in the Yao model of churning and Pareto (a,b) or Exponential (c, d) lifetimes, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.



(a) Recall (b) Precision (c) Average Relative Error

Figure 12: Dataset CAIDA: Recall, Precision and Average Relative Error (mean and confidence interval) varying the failure probability in a fail-stop model of churning, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.



(a) Recall (b) Precision (c) Average Relative Error

Figure 13: Dataset WEBDOCS: Recall, Precision and Average Relative Error (mean and confidence interval) varying the failure probability in a fail-stop model of churning, for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

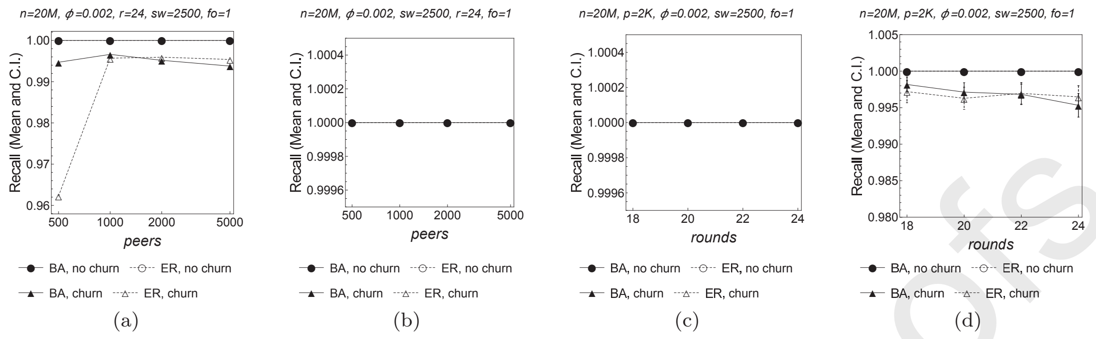


Figure 14: Dataset CAIDA: Recall (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

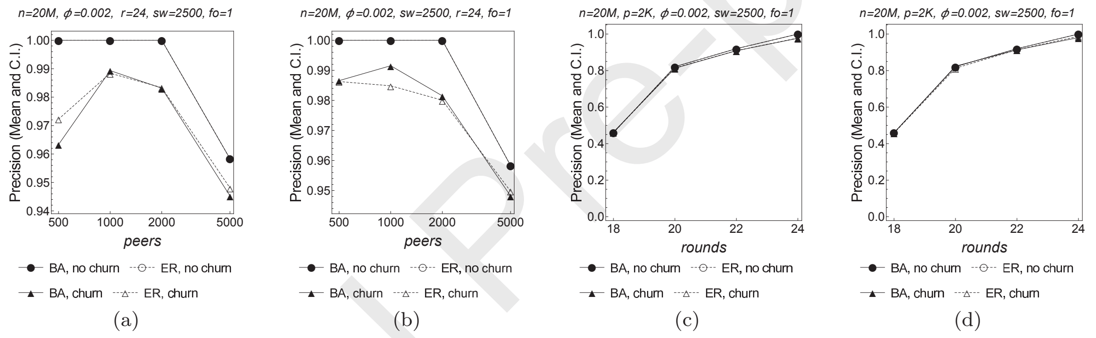


Figure 15: Dataset CAIDA: Precision (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

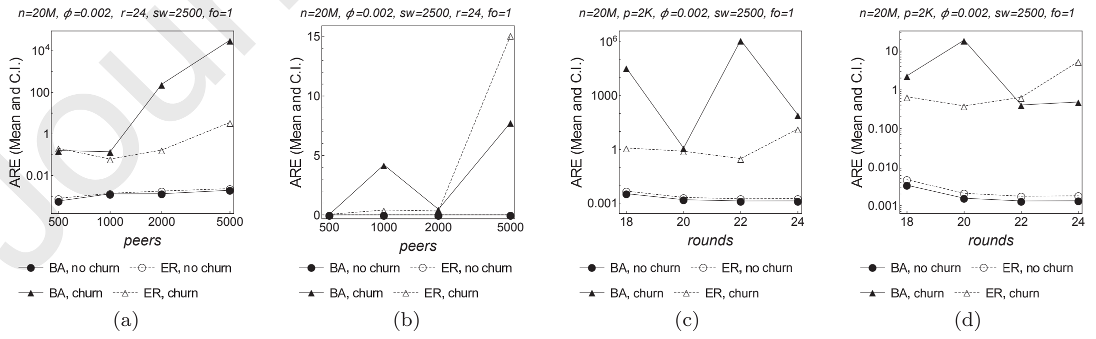


Figure 16: Dataset CAIDA: Average Relative Error (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

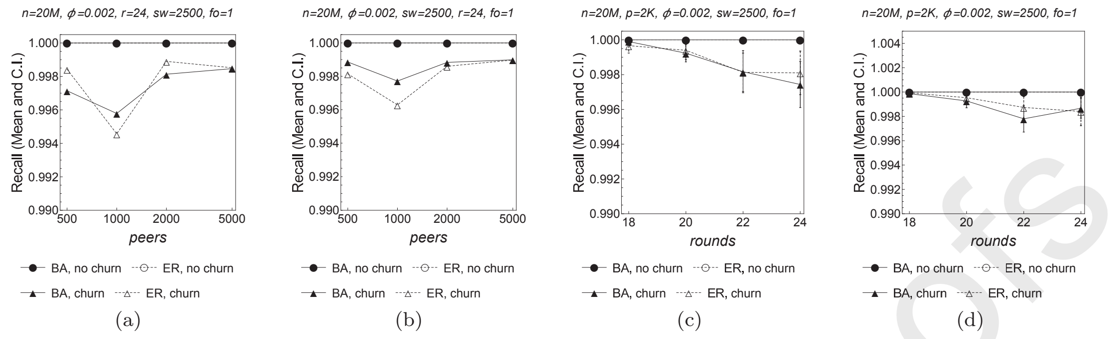


Figure 17: Dataset WEBDOCS: Recall (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

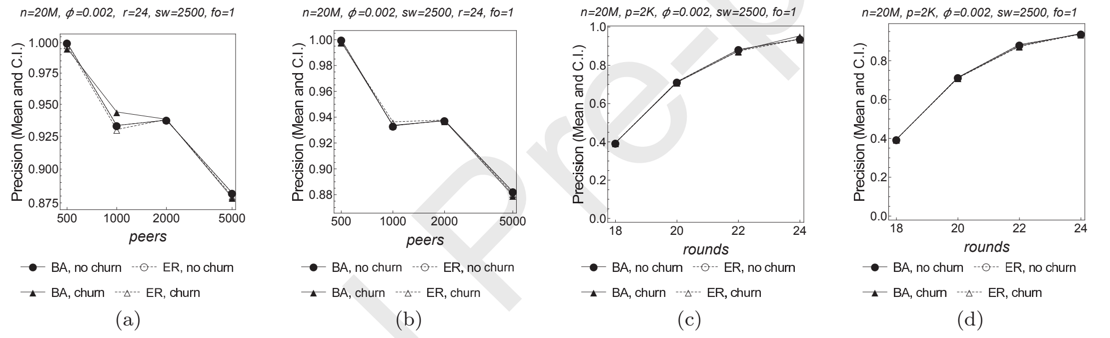


Figure 18: Dataset WEBDOCS: Precision (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

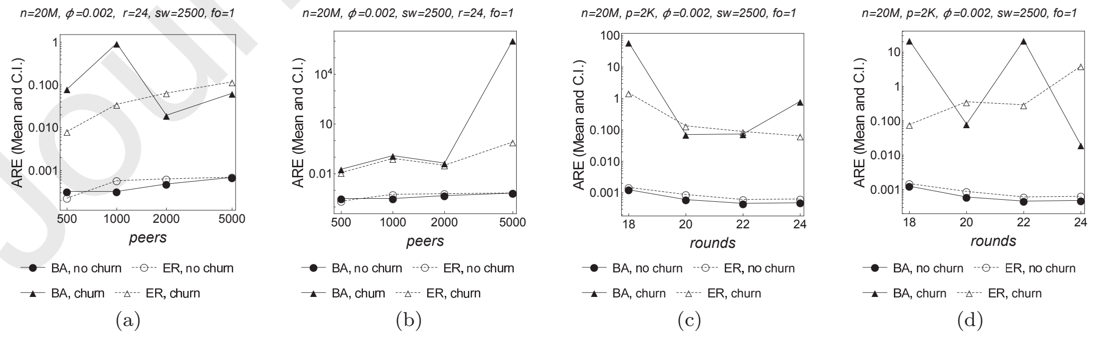


Figure 19: Dataset WEBDOCS: Average Relative Error (mean and confidence interval) varying the number of peers and rounds used in the Yao model of churning and Pareto (a,c) or Exponential (b,d) lifetimes,for both a Barabasi-Albert (BA) and an Erdos-Renyi (ER) type of network graph.

References

[] Cafaro, M., Epicoco, I., Aloisio, G., Pulimeno, M., 2017. Cuda based parallel implementations of space-saving on a gpu. In: 2017 International Conference on High Performance Computing & Simulation (HPCS), Genoa, Italy, 2017. IEEE, pp. 707–714.

[] Cafaro, M., Epicoco, I., Pulimeno, M., 2019. Cmss: Sketching based reliable tracking of large network flows. Future Generation Computer Systems 101, 770 – 784.
URL http://www.sciencedirect.com/science/article/pii/S0167739X1930490X

[] Cafaro, M., Epicoco, I., Pulimeno, M., 2019. Mining frequent items in unstructured p2p networks. Future Generation Computer Systems 95, 1 – 16.
URL http://www.sciencedirect.com/science/article/pii/S0167739X18315838

[] Cafaro, M., Epicoco, I., Pulimeno, M., Aloisio, G., 2017. On frequency estimation and detection of frequent items in time faded streams. IEEE AccessIn press.

[] Cafaro, M., Pulimeno, M., 2016. Merging frequent summaries. In: Proceedings of the 17th Italian Conference on Theoretical Computer Science (ICTCS 2016), Volume 1720. CEUR Proceedings, pp. 280–285.

[] Cafaro, M., Pulimeno, M., Epicoco, I., 2018. Parallel mining of time-faded heavy hitters. Expert Systems with Applications 96, 115 – 128.
URL http://www.sciencedirect.com/science/article/pii/S0957417417307777

[] Cafaro, M., Pulimeno, M., Epicoco, I., Aloisio, G., 2016. Mining frequent items in the time fading model. Information Sciences 370–371, 221–238.

[] Cafaro, M., Pulimeno, M., Epicoco, I., Aloisio, G., 2017. Parallel space saving on multi- and many-core processors. Concurrency and Computation: Practice and Experience 30 (7), e4160–n/a, e4160 cpe.4160.
URL http://dx.doi.org/10.1002/cpe.4160

[] Cafaro, M., Pulimeno, M., Tempesta, P., 2016. A parallel space saving algorithm for frequent items and the hurwitz zeta distribution. Information Sciences 329, 1 – 19.

[] Cafaro, M., Tempesta, P., 2011. Finding frequent items in parallel. Concurrency and Computation: Practice and Experience 23 (15), 1774–1788.

[] CAIDA, 2018. The CAIDA UCSD anonymized internet traces - new york 2018-03-15.
URL http://www.caida.org/data/passive/passive_dataset.xml (accessed February 7, 2019).

[] Charikar, M., Chen, K., Farach-Colton, M., 2002. Finding frequent items in data streams. In: ICALP ’02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming. Springer–Verlag, pp. 693–703.

[] Chen, L., Mei, Q., 2014. Mining frequent items in data stream using time fading model. Information Sciences 257, 54 – 69.

[] Cormode, G., Hadjieleftheriou, M., Aug. 2008. Finding frequent items in data streams. Proc. VLDB Endow. 1 (2), 1530–1541.
URL http://dx.doi.org/10.14778/1454159.1454225

[] Cormode, G., Korn, F., Tirthapura, S., 2008. Exponentially decayed aggregates on data streams. In: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering. ICDE ’08. IEEE Computer Society, Washington, DC, USA, pp. 1379–1381.
URL http://dx.doi.org/10.1109/ICDE.2008.4497562

[] Cormode, G., Muthukrishnan, S., 2005. An improved data stream summary: the count-min sketch and its applications. J. Algorithms 55 (1), 58–75.

[] Cormode, G., Muthukrishnan, S., 2005. What’s hot and what’s not: Tracking most frequent items dynamically. ACM Trans. Database Syst. 30 (1), 249–278.

[] Cormode, G., Shkapenyuk, V., Srivastava, D., Xu, B., 2009. Forward decay: A practical time decay model for streaming systems. In: 2009 IEEE 25th International Conference on Data Engineering. pp. 138–149.

[] Csardi, G., Nepusz, T., 2006. The igraph software package for complex network research. InterJournal Complex Systems, 1695.
URL http://igraph.org

[] Das, S., Antony, S., Agrawal, D., El Abbadi, A., 2009. Thread cooperation in multicore architectures for frequency counting over multiple data streams. Proc. VLDB Endow. 2 (1), 217–228.

[] Datar, M., Gionis, A., Indyk, P., Motwani, R., 2002. Maintaining stream statistics over sliding windows: (extended abstract). In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA ’02. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 635–644.

[] Demaine, E. D., López-Ortiz, A., Munro, J. I., 2002. Frequency estimation of internet packet streams with limited space. In: Proceedings of the 10th Annual European Symposium on Algorithms. ESA ’02. pp. 348–360.

[] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D., 1987. Epidemic algorithms for replicated database maintenance. In: Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing. PODC ’87. ACM, New York, NY, USA, pp. 1–12.
URL http://doi.acm.org/10.1145/41840.41841

[] Epicoco, I., Cafaro, M., Pulimeno, M., Jan. 2018. Fast and accurate mining of correlated heavy hitters. Data Min. Knowl. Discov. 32 (1), 162–186.
URL https://doi.org/10.1007/s10618-017-0526-x

[] Erra, U., Frola, B., 2012. Frequent items mining acceleration exploiting fast parallel sorting on the {GPU}. Procedia Computer Science 9 (0), 86 – 95, proceedings of the International Conference on Computational Science, {ICCS} 2012.

- [] Estan, C., Varghese, G., 2001. New directions in traffic measurement and accounting. In: IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement. ACM, pp. 75–80.
- [] Gelbukhl, A. (Ed.), 2006. Vol. 3878 of Lecture Notes in Computer Science. Springer–Verlag.
- [] Govindaraju, N. K., Raghuvanshi, N., Manocha, D., 2005. Fast and approximate stream mining of quantiles and frequencies using graphics processors. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. SIGMOD '05. ACM, pp. 611–622.
- [] Goyal, A., Daumé, H., 2011. Approximate scalable bounded space sketch for large data nlp. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. EMNLP '11. Association for Computational Linguistics, USA, p. 250–261.
- [] Goyal, A., Daumé, H., Cormode, G., 2012. Sketch algorithms for estimating point queries in nlp. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. EMNLP-CoNLL '12. Association for Computational Linguistics, USA, p. 1093–1103.
- [] Jelasity, M., Montresor, A., Babaoglu, O., Aug. 2005. Gossip-based aggregation in large dynamic networks. ACM Trans. Comput. Syst. 23 (3), 219–252.
  URL http://doi.acm.org/10.1145/1082469.1082470
- [] Jin, C., Qian, W., Sha, C., Yu, J. X., Zhou, A., 2003. Dynamically maintaining frequent items over a data stream. In: Proceedings of the Twelfth International Conference on Information and Knowledge Management. CIKM '03. ACM, New York, NY, USA, pp. 287–294.
  URL http://doi.acm.org/10.1145/956863.956918
- [] Karp, R. M., Shenker, S., Papadimitriou, C. H., 2003. A simple algorithm for finding frequent elements in streams and bags. ACM Trans. Database Syst. 28 (1), 51–55.
- [] Keralapura, R., Cormode, G., Ramamirtham, J., 2006. Communication-efficient distributed monitoring of thresholded counts. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data. SIGMOD '06. ACM, New York, NY, USA, pp. 289–300.
  URL http://doi.acm.org/10.1145/1142473.1142507
- [] Lahiri, B., Mukherjee, A. P., Tirthapura, S., 2016. Identifying correlated heavy-hitters in a two-dimensional data stream. Data Mining and Knowledge Discovery 30 (4), 797–818.
  URL http://dx.doi.org/10.1007/s10618-015-0438-6
- [] Lahiri, B., Tirthapura, S., 2010. Identifying frequent items in a network using gossip. Journal of Parallel and Distributed Computing 70 (12), 1241 – 1253.
  URL http://www.sciencedirect.com/science/article/pii/S0743731510001358
- [] Manjhi, A., Shkapenyuk, V., Dhamdhere, K., Olston, C., 2005. Finding (recently) frequent items in distributed data streams. In: Proceedings of the 21st International Conference on Data Engineering. ICDE '05. IEEE Computer Society, Washington, DC, USA, pp. 767–778.
  URL http://dx.doi.org/10.1109/ICDE.2005.68
- [] Manku, G. S., Motwani, R., 2002. Approximate frequency counts over data streams. In: Proceedings of the 28th International Conference on Very Large Data Bases. VLDB '02. VLDB Endowment, pp. 346–357.
  URL http://dl.acm.org/citation.cfm?id=1287369.1287400
- [] Metwally, A., Agrawal, D., Abbadi, A. E., 2006. An integrated efficient solution for computing frequent and top-k elements in data streams. ACM Trans. Database Syst. 31 (3), 1095–1133.
- [] Misra, J., Gries, D., 1982. Finding repeated elements. Science of Computer Programming 2 (2), 143–152.
- [] Muthukrishnan, S., 2005. Data streams: Algorithms and applications. Foundations and Trends® in Theoretical Computer Science 1 (2), 117–236.
- [] Pulimeno, M., Epicoco, I., Cafaro, M., Melle, C., Aloisio, G., 2018. Parallel mining of correlated heavy hitters. In: Gervasi, O., Murgante, B., Misra, S., Stankova, E., Torre, C. M., Rocha, A. M. A., Taniar, D., Apduhan, B. O., Tarantino, E., Ryu, Y. (Eds.), Computational Science and Its Applications – ICCSA 2018. Springer International Publishing, Cham, pp. 627–641.
- [] Roy, P., Teubner, J., Alonso, G., 2012. Efficient frequent item counting in multi-core hardware. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '12. ACM, pp. 1451–1459.
- [] Sacha, J., Montresor, A., 2013. Identifying frequent items in distributed data sets. Computing 95 (4), 289–307.
- [] Tangwongsan, K., Tirthapura, S., Wu, K.-L., 2014. Parallel streaming frequency-based aggregates. In: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures. SPAA '14. ACM, pp. 236–245.
- [] Wu, S., Lin, H., U, L. H., Gao, Y., Lu, D., 2017. Novel structures for counting frequent items in time decayed streams. World Wide Web 20 (5), 1111–1133.
  URL http://dx.doi.org/10.1007/s11280-017-0433-5
- [] Yao, Z., Leonard, D., Wang, X., Loguinov, D., Nov 2006. Modeling heterogeneous user churn and local resilience of unstructured p2p networks. In: Proceedings of the 2006 IEEE International Conference on Network Protocols. pp. 32–41.
- [] Zhang, Y., 2012. Parallelizing the weighted lossy counting algorithm in high-speed network monitoring. In: Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC),. pp. 757–761.
- [] Zhang, Y., Sun, Y., Zhang, J., Xu, J., Wu, Y., 2014. An efficient framework for parallel and continuous frequent item monitoring. Concurrency and Computation: Practice and Experience 26 (18), 2856–2879.
- [] Çem, E., Öznur Özkasap, 2013. Profid: Practical frequent items discovery in peer-to-peer networks. Future Generation Computer Systems 29 (6), 1544 – 1560, including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems.
  URL http://www.sciencedirect.com/science/article/pii/S0167739X12001859